UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DISEÑO Y DESARROLLO DE UN SISTEMA MULTIMEDIA/VIDEO PARA APOYO DIDÁCTICO A USUARIOS POR CONEXIÓN VÍA MÓDEM A RED UNAM

TESIS

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

PRESENTA:

CARLOS VLADIMIR VILLARROEL CORTÉS

DIRECTOR DE TESIS:

ING. JOSÉ LUIS LEGORRETA GARCÍA

CIUDAD UNIVERSITARIA 1999

Diseño y desarrollo de un sistema multimedia/video para apoyo didáctico a usuarios por conexión vía módem a RedUNAM.

Objetivo: Realizar un sistema multimedia que sirva como herramienta visual de ayuda para los usuarios vía módem de RedUNAM, este sistema deberá ser grabado y distribuido en video-cassette, de esta forma estará al alcance del usuario que no cuente con una máquina de las características multimedia requeridas. Paralelamente, dar a conocer las características de la arquitectura utilizada, que debe ofrecer la mejor relación costo/beneficio. Esto permitirá considerarla como una alternativa útil para fines educativos.

Introducción

La gran influencia del sistema visual de información a distancia más popular, conocido como TV, ha sido el fenómeno social más importante de los últimos años solo comparable a la revolución informática, la reducción de sus costos de producción y la sorprendente emulación de la visión humana en sí misma la convierten en la herramienta más poderosa que jamás haya existido, para formar o deformar la conciencia del individuo.

Estos sistemas visuales de información nunca han estado al alcance de todos, solamente unos cuantos pueden expresar, exponer e imponer sus ideas, sin embargo, indirectamente, los recientes avances en el ámbito de la computación podrían resolver este problema, de modo que el individuo puede dejar de ser un espectador pasivo.

La computadora personal nos permite expresar las ideas con imágenes y sonido de una forma antes imposible, sumado a esto, el auge de las telecomunicaciones digitales que culmina con la red mundial Internet, amplía considerablemente el horizonte de la comunicación para el individuo común y multiplica la utilidad de esta herramienta computacional.

Desafortunadamente, una computadora personal adecuada fabricada por los monopolios informáticos, tiene un precio comparativo mucho mayor que el de un televisor común, además que desde tiempo atrás han sido productos casi desechables que requieren innumerables actualizaciones y por ende, más costos. Esto implica que el problema de la comunicación para todos continúa sin resolverse.

Sin embargo, este trabajo pretende demostrar que existen algunas alternativas, que nos permiten obtener los beneficios de la herramienta llamada computadora personal, sin necesidad de grandes gastos. Además se considera el popular video-cassette como una opción interesante para distribuir las ideas expresadas con un "sello televisivo", mientras que los sistemas de telecomunicaciones digitales reducen sus costos en el ámbito de la transmisión de imágenes en movimiento.

Contenido

Capítulo 1: Antecedentes.

1.1 Internet y RedUNAM en general

Indiscutiblemente, en sus pocos años de existencia, la computadora personal ha cambiado la historia de la raza humana. Pero en los últimos años la fórmula: computadora+telecomunicación *ha acentuado este* cambio en niveles insospechados.

Durante mucho tiempo los medios masivos de transmisión a distancia sólo estuvieron al alcance de unos cuantos. Una red global de computadoras permite una verdadera comunicación a distancia al alcance de muchos.

Si a esto le sumamos el poder de la mejor herramienta procesadora de información que se ha inventado, obtenemos el medio de comunicación del futuro.

RedUNAM es para muchos mexicanos la puerta de entrada a Internet y por lo tanto la herramienta educativa más valiosa en toda la historia de la misma Universidad.

El esfuerzo que ha realizado la UNAM por mantenerse a la vanguardia en esta rama de las telecomunicaciones es digno de ser aplaudido y esperemos que continúe así durante mucho tiempo más.

1.2 Servicio vía módem de RedUNAM

Por ser la forma más económica de conexión a Internet, y considerando que con el tiempo será una necesidad más que un lujo, es posible entender que el número de usuarios aumentará considerablemente en los próximos años.

El servicio vía módem de RedUNAM cuenta con un Departamento especializado en atender las dudas de todos los usuarios vía módem, generalmente estos usuarios saben muy poco de computación y la mayoría de sus problemas son realmente fáciles de solucionar. Pero de una u otra forma, este servicio requiere tiempo y dinero. La infraestructura de RedUNAM crece día con día debido a la demanda del servicio y con ésta también crece el número de problemas.

En este sentido, es importante cualquier tipo de ayuda externa para los usuarios, por lo menos en los aspectos concernientes al uso y configuración básica de los programas, para de esto modo minimizar el número de problemas a resolver.

1.3 Necesidad de una herramienta audio-visual

Es por todo mundo reconocido el hecho de que una imagen vale más que mil palabras.

La mayoría de los usuarios tienen problemas con la instalación y el uso del software de conexión a Internet, debido a la falta de experiencia visual con el manejo del mismo, así que una herramienta audio-visual será de gran ayuda.

1.4 Métodos de solución

No hay que perder de vista la descripción del problema:

Se requiere de una herramienta didáctica como apoyo para los usuarios de acceso *vía módem a* RedUNAM. Existen manuales en forma de escritos y libros, pero esto parece no ser suficiente para muchos usuarios además de otras razones ya expuestas.

Así que una posible solución es realizar un sistema de presentación de información de forma tal que el usuario pueda observar los procesos de configuración y uso de los programas concernientes a Internet, así como una introducción a los conceptos relacionados con Internet mismo.

Desde el punto de vista de la Ingeniería, el problema se debe resolver encontrando la solución más adecuada, es decir óptima.

1.4.1 Video-cinta

Es posible hacer una grabación de la configuración de cada uno de los programas involucrados con ayuda de voz de tal forma que sea lo más cercano a una asesoría personalizada para el usuario, en este sentido, el usuario obtendrá una experiencia audio-visual que puede ser determinante para un adecuado y más rápido aprendizaje. Casi toda la gente cuenta con un una "videocasetera", esta es la alternativa más económica para la mayoría de los usuarios.

1.4.2 Tutorial Multimedia en forma de software

Esta debería ser la opción ideal, desafortunadamente no todos los usuarios cuentan con una máquina de características multimedia aceptables, además la cantidad de información es demasiada como para distribuirse en discos flexibles. Se requiere un CD ROM como solución ideal, pero esto implica desde luego más costos que el de un videocassette. (Finalmente esta opción no se descartará)

1.4.3 Tutorial en Internet

Además de las limitantes actuales en cuanto a velocidad y manejo masivo de gráficos, actualmente más bien convendría colocar en Internet las páginas digitalizadas de un manual en papel, lo cual no resolvería el problema principal, además no hay que perder de vista que el objetivo es auxiliar al usuario que apenas intenta acceder a Internet.

1.5 Realización de una video-cinta

La video-cinta se ha vuelto tan común que difícilmente se puede encontrar a alguna persona que no tenga acceso a un aparato reproductor de este tipo. De este modo, simplemente por economía, es la opción más adecuada.

1.6 Justificación

Si comparamos costos entre la impresión de un manual en papel y la copia de un videocassette, obtenemos los siguientes resultados:

» Manual en papel

Requerimientos	Costo estimado
100 copias a $0.15 c/u	$15.00
Pastas	$ 2.00
Engargolado (incluyendo material)	$ 4.00
» Total:	$21.00

» Videocassette*

Requerimientos	Costo estimado
videocassette de 2 horas	$12.00
copiado	$ 2.00
etiquetado	$ 1.00
» Total:	$15.00

* Conviene además comparar el grado de automatización que conlleva la copia de un videocassette o un disco con el tiempo y la necesidad de maquinaria para duplicar un libro. Además del hecho de que cada día aumenta el número de usuarios.

Con la ayuda de una herramienta audio-visual, la necesidad de un equipo de atención a usuarios se reduciría, pudiendo enfocar más la atención del mismo a la investigación y experimentación con otras y nuevas tecnologías.

1.7 Puntos importantes a tratar en la secuencia audio-visual

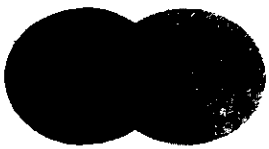
-Introducción básica a Internet/RedUNAM
-Instalación del software
-Configuración y uso del programa de conexión a Internet/RedUNAM
-Configuración y uso de los programas clientes básicos:
    Telnet
    Correo electrónico
    FTP
    WWW
-Solución de posibles problemas de configuración

Capítulo 2: Planteamientos y metodología básica para la realización de una secuencia de video

2.1 Características de la señal de video

2.1.1 RGB o los colores básicos

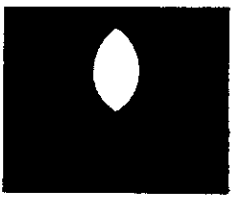
Fig.1

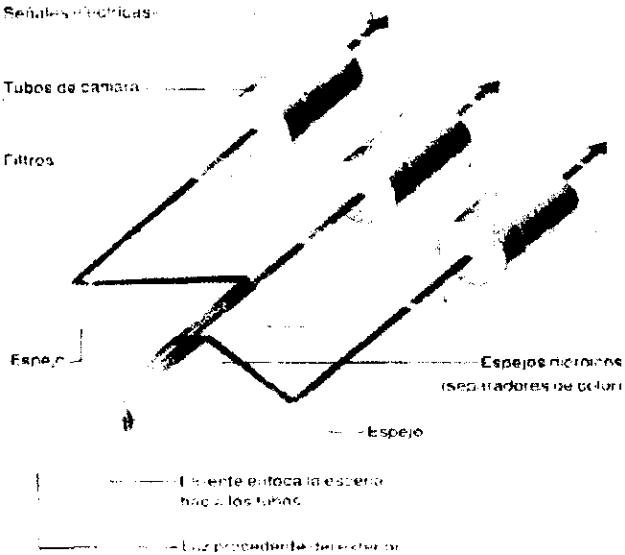


Mucho antes de que los mecanismos físicos del ojo humano fueran entendidos ya se sabía que una combinación de los colores básicos Rojo Verde y Azul nos permiten formar cualquier color visible por el ser humano. La mezcla de colores sustractiva utilizada en las artes gráficas es el más claro indicio de esto.

Inclusive organizaciones como la Comisión Internacional de Iluminación ya habían realizado algunos intentos para cuantificar de forma precisa las percepciones de color del ojo humano.

Fig.2,3





En este sentido la televisión de color se basó en una combinación aditiva de los tres colores básicos.

Hoy sabemos que los receptores de luz de la retina humana cuentan con tres pigmentos sensitivos responsables de la detección de los mencionados colores básicos. Pero es interesante saber que estos tres pigmentos no son de ninguna manera un estándar para todos los seres vivos, pues existen algunas especies de camarones que tienen hasta 20 pigmentos fotoreceptores diferentes.

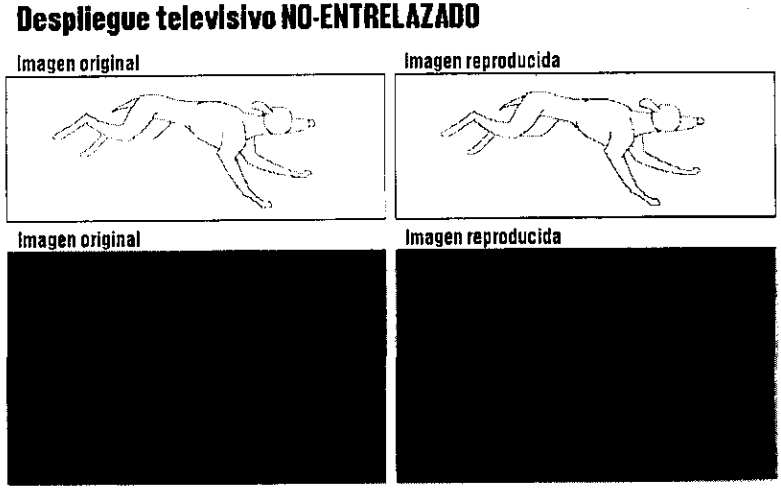
2.1.2 Persistencia visual

Fig.4

El ojo humano retiene una imagen durante una fracción de segundo después de que esta ha desaparecido. Esta propiedad llamada persistencia de la visión es esencial para todas las tecnologías de despliegue visual, la idea básica es simple, una serie de cuadros estáticos son presentados rápidamente y de esta forma que se crea la ilusión de movimiento.

El parámetro a seguir fueron los 24 cuadros por segundo utilizados en el cine, de este modo en Europa, donde la frecuencia de la línea de alimentación son 50Hz, se utilizó un despliegue de 25 cuadros por segundo para simplificar algunos circuitos de video. Asimismo en América se optó por los 30 cuadros por segundo, puesto que la línea de alimentación tiene una frecuencia de 60Hz.

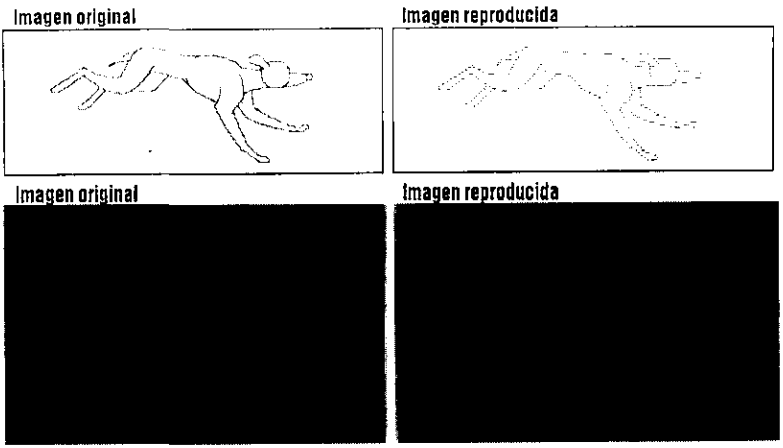
Fig.5



30 cuadros por segundo parecerían más que suficientes para cualquier secuencia de video reproducida, pero debido a las limitantes técnicas en la velocidad del barrido vertical, entre más saturada y compleja sea la secuencia de imágenes, parecería que se minimiza el tiempo de persistencia de la visión y aparece un parpadeo desagradable, así que imágenes de este tipo requieren una mayor frecuencia de repetición para evitar este problema.

Fig.6

Despliegue televisivo ENTRELAZADO



En este sentido se utilizó un sistema de exploración de imagen conocido como entrelazado y que consiste en doblar el número de repeticiones presentando primero las líneas impares de un cuadro y luego las líneas pares, de este modo, en realidad se presentan 60 cuadros por segundo (50 en Europa) a costa de una menor resolución vertical, la mitad para ser exactos.

Es interesante observar que a pesar de la baja en resolución, en muchos casos esto no es notable aún con imágenes en movimiento rápido, sobre todo en despliegues o animaciones generadas digitalmente o por computadora que permiten un alto nivel de sincronía entre cuadros. Por supuesto en algunos casos, sobre todo en video generado analógicamente con rápidos movimientos horizontales o verticales, el entrelazado crea algunos efectos extraños que a pesar de todo no son inaceptables.

2.1.3 Exploración de la imagen

2.1.3.1 Sistemas de TV utilizados en el mundo

Actualmente se utilizan 3 principales sistemas de televisión:

NTSC (*National Television Systems Committee*) - América y Japón.
PAL (*Phase Alternation Line rate*) - Europa.
SECAM (*Sequential Couleur avec Memoire*) - Francia/Ex Unión Soviética.

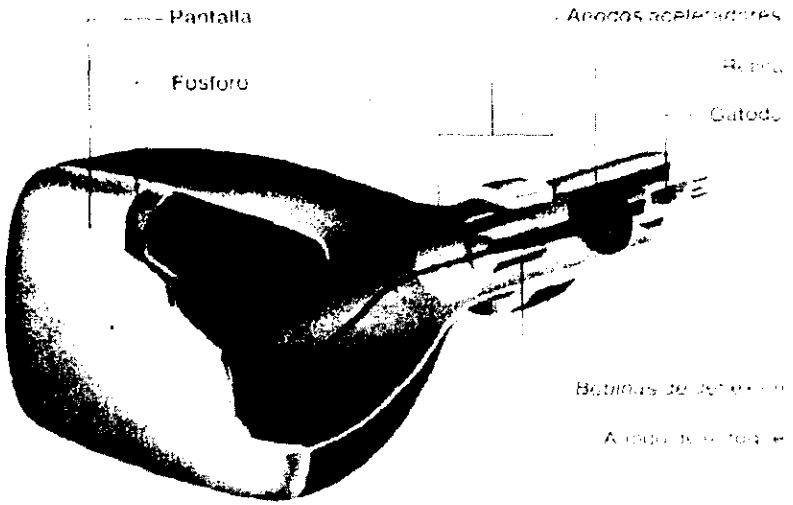
	Líneas	Líneas activas	Resolución vertical	Aspecto	Resolución horizontal	Cuadros por seg
NTSC	525	484	242	4/3	427	29.97*
PAL	625	575	290	4/3	425	25
SECAM	625	575	290	4/3	465	25

*Cabe hacer notar que el formato NTSC utiliza ligeramente menos de 30 cuadros (29.97) desde que se incluyó la información de color, esto para que toda la señal moderna de TV se encontrara dentro de los límites del ancho de banda ya definido para las señales de video y audio.

Como se observa, entre menos cuadros se puede obtener más resolución, pero además existen otras diferencias entre estos sistemas: PAL utiliza un formato especial de alternancia entre líneas para disminuir errores de color durante la transmisión y SECAM difiere fuertemente de PAL y NTSC en la forma de modular las señales. Nos enfocaremos al formato NTSC puesto que es el que utilizaremos durante el desarrollo de este trabajo y de cualquier modo, el principio básico de exploración de la imagen es el mismo para los tres sistemas.

2.1.3.2 Tubo de rayos catódicos

Fig.7



El tubo de rayos catódicos, inventado en 1875 por William Crookes y perfeccionado con posterioridad por varios científicos es uno de los elementos básicos para los despliegues de video actuales: monitores y tv. Consiste en una ampolla de vidrio en cuyo interior se ha forzado el vacío y en uno de sus extremos se encuentra situado un sistema denominado cañón electrónico, capaz de crear un fino haz de rayos catódicos; esto es, un "chorro" de electrones que viajan desde el polo negativo al positivo a muy alta velocidad.

El cañón electrónico consiste en un cilindro metálico -CATODO- conectado al polo negativo de un generador eléctrico. En dicho cátodo, gracias a la acción de un filamento, se produce un aumento de temperatura que tiene como consecuencia la emisión por parte del mismo de electrones. Estos electrones emitidos por el cátodo son atraídos por una serie de anillos metálicos conectados al polo positivo del generador que comunican a éstos una gran velocidad a la vez que los concentran en un fino haz que es lanzado hacia el extremo del tubo que tiene la forma más ancha.

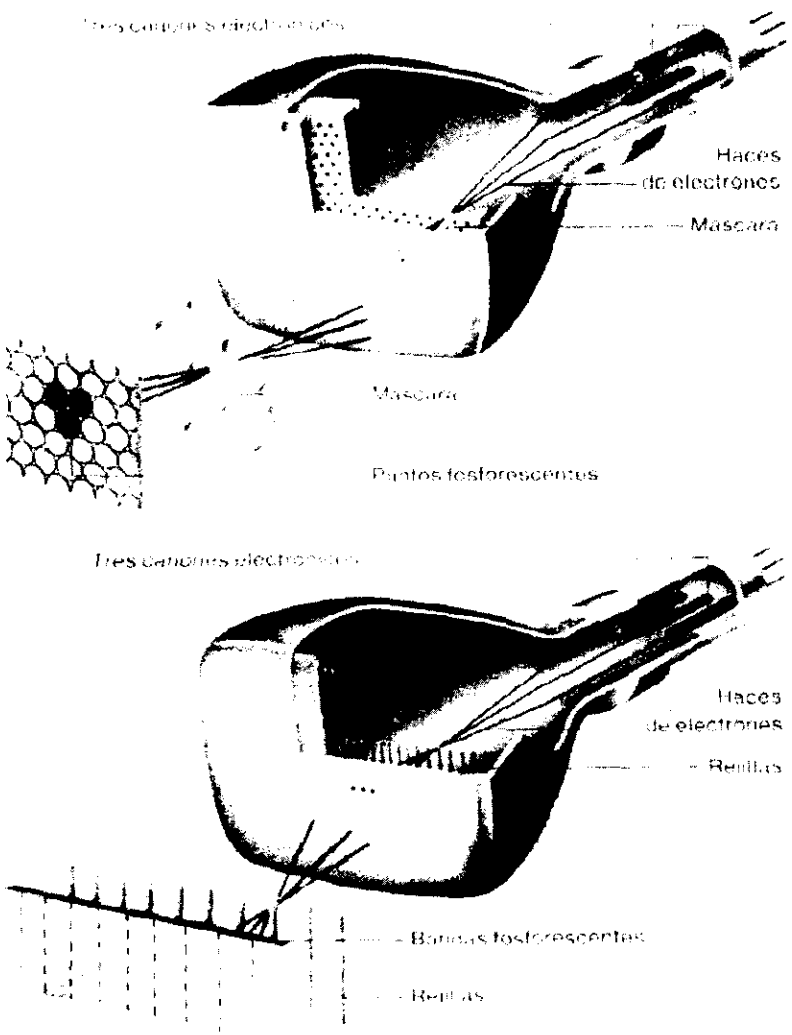
Debido a la carga eléctrica que poseen los electrones pueden ser atraídos tanto por fuerzas eléctricas como magnéticas, por lo que el conjunto del haz electrónico puede ser desviado por medio de unas bobinas que se colocan en el cuello del tubo entre el cañón electrónico y la pantalla. Estas bobinas al ser recorridas por una corriente eléctrica, crean una fuerza magnética cuya intensidad depende del valor de la misma y que, por tanto, permite graduar la desviación del haz electrónico.

Actuando sobre dos pares de bobinas, uno colocado de forma vertical y otro transversal, se puede desviar el haz en cualquier dirección, permitiendo que alcance cualquier punto de la pantalla en el cual quedará reflejada su llegada gracias a una sustancia -FOSFORO- que recubre el interior de la misma y que tiene la propiedad de volverse luminosa al incidir sobre ella los electrones. Gracias a esta propiedad,

en el punto de la pantalla sobre la que incide el haz aparece un punto luminoso que se transforma en una *línea cuando el haz recorre la pantalla* con una cierta velocidad.

Un sistema como el que se detalló aquí permite obtener una imagen monocromática con una gama de intensidades que van del negro al color más brillante de la sustancia luminiscente en el interior de la pantalla.

Fig.8,9



En este sentido, las pantallas de color son *similares a las pantallas monocromáticas*, pero se diferencian en sus tres cañones de electrones, uno para cada color básico (rojo, verde y azul), y en su pantalla *especial sobre cuya parte interior* se distribuyen tres sustancias luminiscentes diferentes, que dan luz roja, verde o azul al incidir los electrones en ellas. Para conseguir que los electrones procedentes del cañón rojo solo incidan en la sustancia luminiscente roja, y que ocurra lo mismo con los otros colores,

existen varias soluciones, origen de otros tantos tubos de color. Los más usados en la actualidad son el de máscara perforada y el de rejillas verticales.

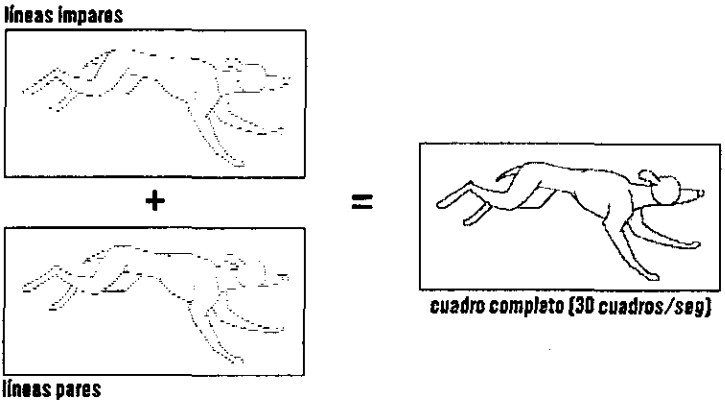
El primero intercala una máscara metálica llena de pequeñas perforaciones entre los cañones de electrones y la pantalla. Según un ángulo de procedencia, los electrones, al pasar a través de las perforaciones, incidirán en un punto diferente de la pantalla donde se han depositado las sustancias luminiscentes correspondientes. La imagen así formada consta de pequeños tríos de puntos, cada uno de un color; mezclados en el ojo humano darán como resultado la visión de uno solo, mezcla aditiva de los tres colores fundamentales. Como se observa, este tubo de color requiere un alineamiento perfecto entre cada cañón de electrones, las perforaciones de la máscara y las sustancias luminiscentes del color correspondiente.

El tubo de imagen de rejillas verticales tiene las sustancias luminiscentes dispuestas en forma de bandas verticales. Una rejilla formada por hilos conductores, situada detrás de la pantalla, concentra los electrones procedentes de los cañones en cada una de las bandas de color, según su procedencia. La imagen, igual que en el tubo de máscara, está formada por tríos de puntos, pero esta vez dispuestos en línea recta y no en triángulos, simplificando el enfoque de cada haz de electrones.

2.1.3.3 Señales de desviación para la exploración de la imagen

Fig.10

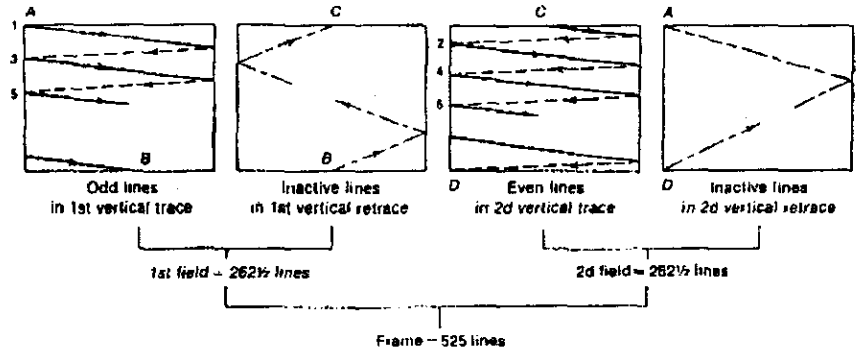
Exploración de una imagen en televisión



La forma en que se explora una imagen para formar un cuadro de televisión se conoce como sistema de barrido lineal uniforme. En éste, la cámara explora la imagen en líneas sucesivas a una velocidad constante y el receptor la reconstruye de la misma manera y a la misma velocidad.

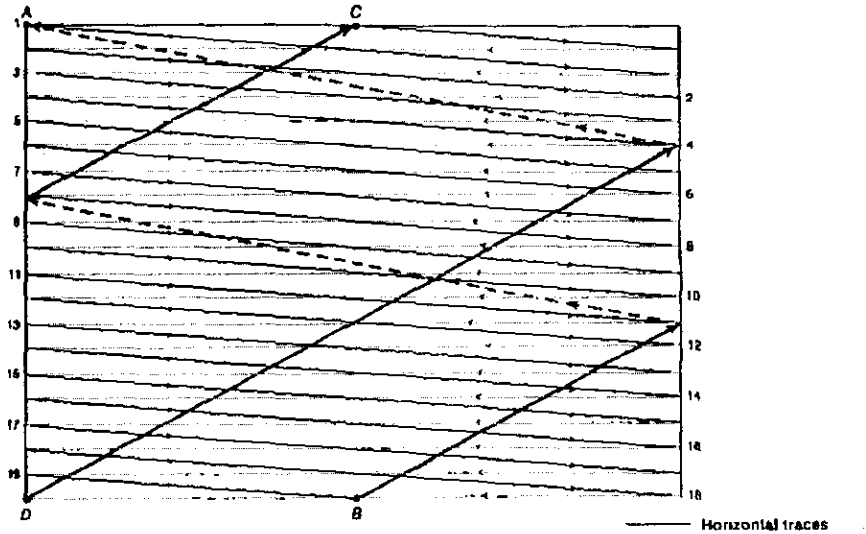
En un monitor ó TV un cuadro de imagen está formado por una serie de pequeños puntos de diferente intensidad conocidos como elementos de imagen o pixels. La exploración es entrelazada como ya se mencionó, es decir, se explora dos veces por cada cuadro.

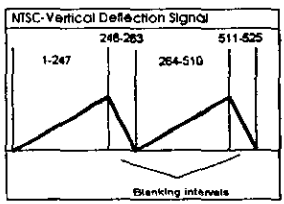
Fig.11



La información siempre es desplegada de izquierda a derecha. Después de que cada línea es escrita, cuando el cañón regresa otra vez a la izquierda, la información es eliminada. Asimismo cuando la señal alcanza la parte baja de la pantalla también la información es eliminada hasta que regresa a la parte alta para escribir la siguiente línea.

Fig.12,13





El despliegue convencional NTSC tiene 525 líneas verticales. De cualquier modo, las líneas 248 a 263 y 511 a 525 típicamente son eliminadas (*blanked*) para dar tiempo a que el cañón de electrones regrese a la posición superior izquierda y poder comenzar con la exploración del siguiente cuadro. Puede notarse que el cañón no regresa directamente a la parte alta, sino que sigzaguea un poco.

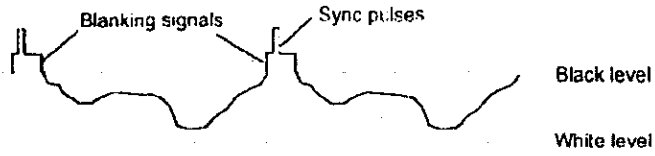
La señal de exploración vertical es simplemente una rampa positiva hasta el momento de regreso a la esquina superior izquierda. En ese momento, durante las líneas eliminadas (*vertical blanking*), se convierte en una rampa negativa. Su frecuencia es de 29.97 Hz. en el formato NTSC.

La señal de exploración horizontal es muy parecida y su frecuencia es de 525 líneas por 29.97 campos por segundo, es decir 15,734 Hz. Esto implica que aproximadamente 63.6 uS son utilizados por línea y alrededor de 10 uS de estos se utilizan en el intervalo de eliminación de señal de línea (*horizontal blanking*), durante el regreso del cañón de electrones para posicionarse al inicio de la línea siguiente.

2.1.4 La señal compuesta de video

2.1.4.1 Video compuesto

Fig.14



El haz de electrones es modulado conforme atraviesa la línea horizontal y esta modulación produce cambios de intensidad en el haz electrónico, que a su vez genera distintos niveles de intensidad monocromática en la pantalla. La velocidad de las señales y los mecanismos de desviación del cañón que produce el haz electrónico permiten que el ojo vea cuadros completos y no solo puntos en la pantalla.

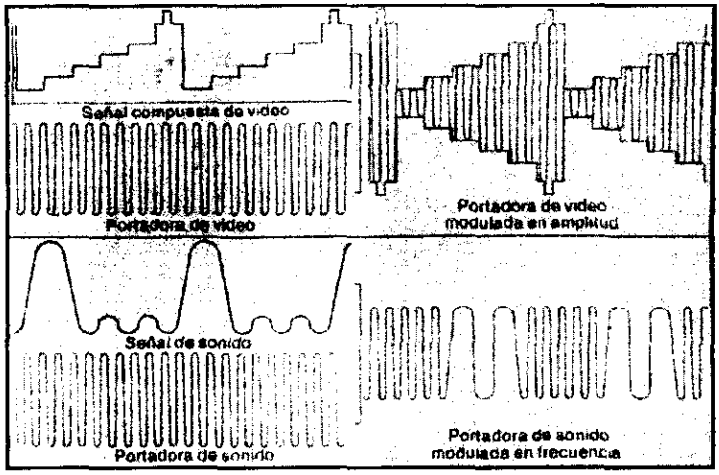
Pero el recomponer una imagen exige una correspondencia muy precisa entre emisor y receptor para recibir los mismos cuadros que han sido grabados y enviados, por este motivo, los intervalos de eliminación o borrado de señal (*blanking*) son utilizados para insertar señales o pulsos de sincronía relacionados con cada línea y cada cuadro.

En la fase de captación o generación de imagen ya sea mediante una videocámara o una computadora es fundamental el denominado generador de señales de sincronismo, un circuito electrónico que origina los impulsos de tensión que, superpuestos a las señales de luminosidad, marcarán el inicio de la lectura de las diferentes líneas y de las imágenes sucesivas en el receptor. La señal de imagen junto con la de sincronismo y las de borrado de los movimientos de retroceso horizontal y vertical de haz de electrones sobre la pantalla, es lo que se denomina señal compuesta de video o señal de video compuesto.

Es interesante saber que casi una tercera parte del total de la amplitud de la señal de video compuesto se utiliza únicamente para información de sincronía.

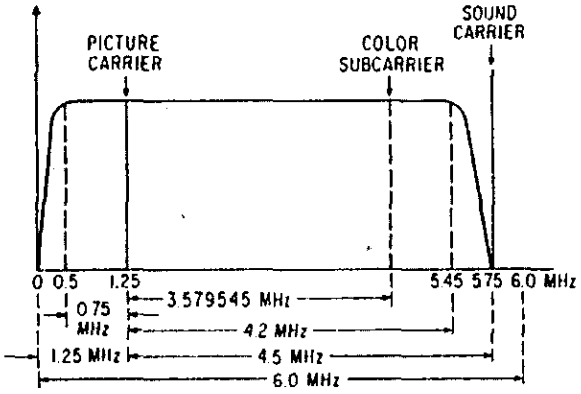
2.1.4.2 Algunos tópicos interesantes de la señal de TV analógica

Fig.15



Típicamente la señal de video utilizada en la TV es emitida, ya sea por microondas o cable coaxial, utilizando una alta frecuencia portadora modulada en amplitud para mantener su calidad y prolongar el alcance. Por otra parte la señal de audio de la TV es modulada en frecuencia para reducir interferencias y distorsión. Sin embargo, para aplicaciones que no requieren transferencias a grandes distancias, en donde inclusive la señal de video es separada de la de audio, esta modulación en amplitud no es necesaria.

Fig.16



Desde un principio el ancho de banda para la señal de TV fue y sigue siendo actualmente de 6Mhz, aunque existe un espacio de 1.25Mhz en el extremo bajo y otro de 0.25Mhz en el extremo más alto para evitar interferencias de tipo *cross-talk* con otros canales.

La TV en "blanco y negro" o monocromática recibe una sola señal de video que corresponde a la información de blancos, grises y negros, conocida como luminancia (Y). Se puede decir que esta señal contiene la información de brillo y detalle. Considerando que oficialmente esta señal utiliza los primeros 4Mhz, en la práctica 3.2Mhz para evitar interferencias, del total de 6Mhz, entonces podemos preguntarnos ¿Cómo se añadió la información de color para mantener la compatibilidad con los receptores monocromáticos?

La TV en color es capaz de captar 3 señales de luminancia distintas una para cada color básico (RGB). Si estas se enviaran de la misma forma que la señal monocromática se requeriría un mayor ancho de banda y además esta información sería incompatible con los receptores monocromáticos. Para evitar esto se crearon dos nuevas señales (Q e I) que sumadas forman la información de crominancia. Pero estas dos señales tienen características especiales que permiten añadirlas a la información de luminancia (Y) sin necesidad de aumentar el ancho de banda total de 6Mhz.

Una de estas señales (Q) contiene los tintes verde-púrpura y la otra (I) los tintes naranja-cian. Si consideramos que el ojo humano es más sensible a las variaciones espaciales en el naranja-cian que para el verde-púrpura entonces podemos reducir el ancho de banda requerido para éste último. Por esta razón el naranja-cian (I) tiene un ancho de banda máximo de 1.5MHz y el verde-púrpura (Q) un ancho de banda máximo de 0.5Mhz.

Estas dos señales, ambas moduladas por una onda portadora también llamada subportadora de color, con una frecuencia de 3.58Mhz, son moduladas en fase y por esto la nomenclatura de ambas señales ayuda a recordar que sucede con estas. La señal I está en fase con la portadora y la señal Q está en cuadratura (90 grados de defasamiento) con respecto a la misma portadora. Estas dos señales se suman para formar la señal de crominancia (C), esta señal de crominancia tiene dos propiedades interesantes: la magnitud de la señal representa la saturación de color y la fase representa el tinte.

De este modo los televisores en blanco y negro pueden utilizar la información de luminancia normalmente y los televisores en color pueden obtener a partir de la información de crominancia (tinte y saturación) los colores rojo y azul, posteriormente la señal del color verde se obtiene restando a la señal de luminancia la roja y la azul.

Es así como se aprovecha el ancho de banda de 6Mhz que posiblemente siga siendo utilizado incluso para la moderna TV de alta definición, solo que esta vez las señales de imagen y audio serán una secuencia de datos digitales ya comprimidos (MPEG) y por lo tanto estos métodos analógicos de "compresión" ya no serán muy utilizados.

2.1.4.3 Componentes de la señal de video compuesto

Fig.17

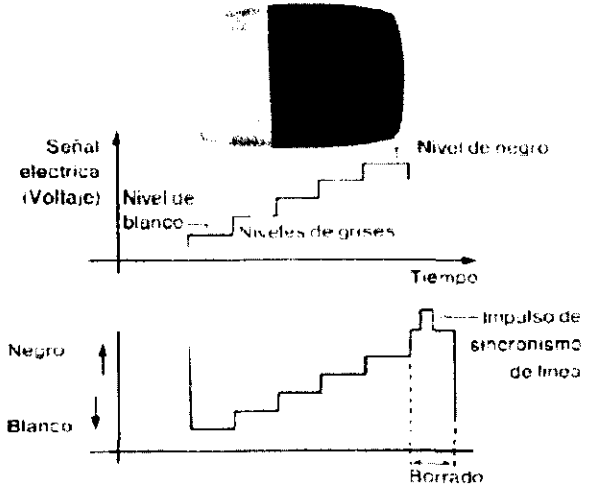
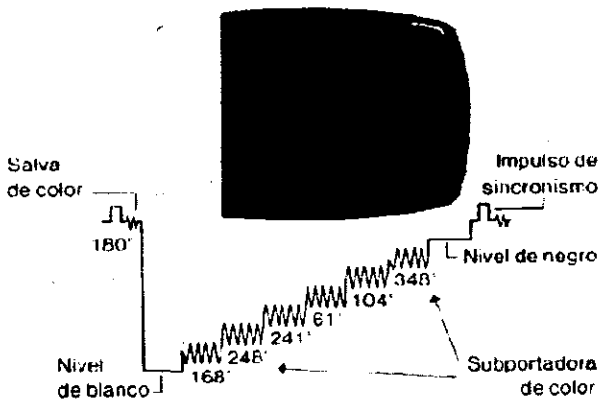


Fig.18



Los siete elementos de la señal de video compuesto son:

1)Pulso de sincronía de línea o sincronía horizontal
2)Ráfaga de referencia de color o salva de color (*burst*)
3)*Referencia del nivel de negro (setup)*
4)Información de luminancia de la imagen
5)Información de saturación del color (color)
6)Información de tinte del color (*hue*)
7)Pulso de sincronía de cuadro o sincronía vertical

Pulso de sincronía de línea o sincronía horizontal

Una vez que el haz de electrones ha regresado a la izquierda y se encuentra en la posición inicial de la nueva línea, los pulsos de sincronía horizontal indican el momento de iniciar el barrido de ésta, de tal modo que las líneas en el receptor se reproduzcan acordes con el emisor.

Ráfaga de referencia de color o salva de color (*burst*)

Para asegurar un tinte y saturación de color estándar, una ráfaga de referencia de color a 3.58Mhz es agregada antes de la información de imagen en cada línea de exploración. Esta es una señal seno de 9 ciclos.

Referencia del nivel de negro (*setup*)

El nivel de negro también llamado inicial o pedestal es definido como 7.5 IEEE unidades (Instituto de Ingenieros Eléctricos y Electrónicos). En un principio estas unidades fueron conocidas como unidades IRE del Instituto de Ingenieros en Radio.

Información de luminancia de la imagen

La luminancia de la imagen varía de 7.5 IEEE unidades para el nivel de negro hasta 100 IEEE unidades para el valor pico o nivel de blanco.

Información de saturación del color (*color*)

La saturación de color es superpuesta en la información de luminancia de la imagen. Esta superposición es llamada la subportadora de 3.58MHz. Esta es comparada con la ráfaga estándar de referencia de color y esta comparación determina la saturación de los colores desplegados.

Información de tinte del color (*hue*)

El tinte está también presente en la subportadora de 3.58Mhz. La exactitud de los colores en la imagen está determinada por las variaciones de fase o rotación de esta señal con respecto a la ráfaga de referencia o salva de color.

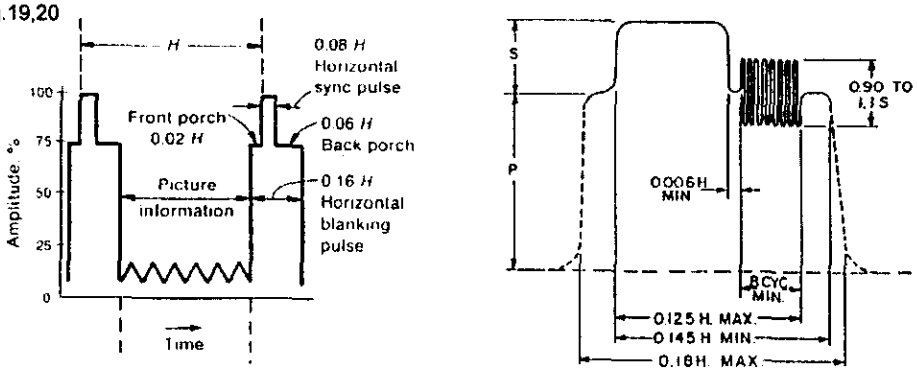
Pulso de sincronía de cuadro o sincronía vertical

Al final de cada campo o cuadro de información, el haz de electrones regresa a la parte superior de la pantalla y espera, el pulso de sincronía vertical indica el momento de inicio del nuevo campo tomando en cuenta el entrelazado, de este modo se asegura una imagen estable.

2.1.4.4 Características y tópicos interesantes de los intervalos de borrado para el trabajo con máquinas de video analógico

2.1.4.4.1 Intervalo de borrado horizontal (*horizontal blanking*)

Fig.19,20



H=Distancia entre pulsos (*distance between pulses*)

El intervalo de eliminación o borrado horizontal (*horizontal blanking*), también conocido como pulso o señal de borrado horizontal ocurre entre en final de la exploración de una línea y el inicio de la siguiente. Dentro de este intervalo de borrado se encuentra el pulso de sincronía horizontal y también la ráfaga de referencia de color si la señal contiene información de crominancia.

Pórtico de entrada (*front porch*): Este tiempo se utiliza para permitir que el haz de electrones anulado (*blanked*) regrese al lado izquierdo de la pantalla y así poder comenzar con el trazo de la siguiente línea.

Filo de entrada del pulso de sincronía (*leading edge*): Sincroniza el barrido de la nueva línea entre el emisor y el receptor. En las videograbadoras este filo se usa para identificar electrónicamente el inicio del intervalo de borrado horizontal.

Nivel de sincronía horizontal: En algunas máquinas se utiliza este nivel como referencia para establecer correctamente el nivel de corriente directa de la señal de TV.

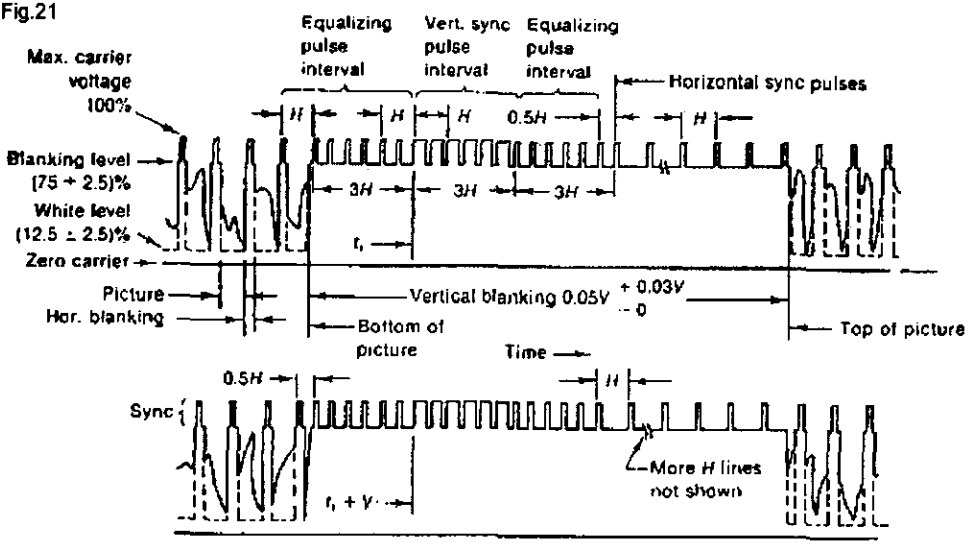
Filo de salida del pulso de sincronía (*treading edge*): Se usa en la grabación de video para identificar el inicio del pórtico de salida. Esto es con el propósito de disparar adecuadamente los circuitos que controlan el *burst*.

Sincronía de color o ráfaga (*burst*): Esta señal da una referencia de amplitud y fase constantes para que el receptor pueda decodificar correctamente las variaciones de fase y amplitud de la información de color que contiene la línea en cuestión.

Pórtico de salida (*back porch*): Durante este tiempo se genera una señal sin información de imagen (*blank signal*, con o sin *burst*) en la pantalla de TV, hasta el inicio de la información de línea.

2.1.4.4.2 Intervalo de borrado vertical (*vertical blanking*)

Fig.21



Desde el inicio del sistema de TV el intervalo de borrado vertical ha mantenido una relación con la frecuencia de la línea de alimentación, pero al introducirse la información de color la frecuencia de 60Hz se tuvo que reducir ligeramente como ya se explicó.

El principal objetivo de la información del intervalo de borrado vertical es mantener una perfecta sincronía entre el dispositivo que está generando la imagen y el que la está reproduciendo mediante la exploración vertical. También dentro de este intervalo encontramos la información que asegura que los dos campos se entrelacen y no se encimen.

La parte más importante del intervalo de borrado vertical es el periodo de 9 líneas formado por los pulsos preigualadores, los pulsos de sincronía vertical o aserraciones y los pulsos postigualadores. Además de esto, el intervalo vertical, contiene de 9 a 12 líneas sin información de imagen que se utilizan en los sistemas de transmisión para insertar señales de información activa de otro tipo.

Pulsos preigualadores (*Equalizing Pulse*): Se encuentran al inicio del intervalo de borrado vertical y son seis. Estos pulsos ocupan un periodo equivalente al de 3 líneas de TV y la manera en que mantienen el entrelazado está determinada por el hecho de que a pesar de ser siempre seis pulsos, la relación de tiempo que presentan con el resto de la señal difiere para cada campo.

Pulsos de sincronía vertical (*Vertical Sync Pulse*): La porción de sincronía vertical que está comprendida dentro del intervalo de borrado vertical se conoce como pulsos de aserración, pues está dividida en seis partes o aserraciones iguales. La sincronía vertical tiene el promedio de valor negativo más alto de la señal de video compuesto. Esto se usa para generar un pulso que se utiliza como referencia vertical del sistema.

Pulsos postigualadores (*Equalizing Pulse*): Estos seis pulsos son idénticos a los preigualadores y en el sistema de TV actual no tienen ninguna función específica.

Pórtico de salida: La parte del intervalo vertical a partir del periodo de 9 líneas hasta la primera línea que contiene información que forma parte de la imagen se conoce como pórtico de salida y contiene el número necesario de pulsos de sincronía horizontal (líneas) para completar el campo.

En estas 12 líneas sin información activa, correspondientes a los pulsos postigualadores (3) y al pórtico de salida (~9), se pueden insertar algunas señales de prueba que sirven para comprobar de una manera continua y confiable la calidad de la señal tanto en grabación como en transmisión o para colocar en ellas algún tipo de información relativa a la imagen grabada pero que no forma parte de esta, como por ejemplo señales de prueba o referencias para mantener la fidelidad de una transmisión en cuanto a los niveles de luminancia, croma, el nivel de negro e inclusive la fase del burst.

- Identificación de campos para el entrelazado

Como ya se mencionó, cada cuadro de TV está formado por 2 campos entrelazados. Tenemos un intervalo vertical asociado a cada campo, cada uno de los cuales contiene el mismo tipo de información, el mismo número de pulsos y su duración es casi la misma, pero hay entre ellos algunas diferencias que nos permiten identificar al campo UNO y al campo DOS.

Esta identificación es muy importante tanto para asegurar el entrelazado correcto como para los sistemas de edición ya que requieren de esta información para su correcto funcionamiento.

Si consideramos al intervalo de borrado vertical como el principio del campo y comparamos el del campo UNO con el del campo DOS encontraremos 6 diferencias básicas:

1. El principio del campo UNO está precedido por una línea de video completa; el principio del campo DOS está precedido por media línea de video.

2. El filo de entrada del pulso de sincronía vertical del campo UNO coincide en tiempo con los filos de entrada de los pulsos de sincronía horizontal; para el campo DOS este filo queda a la mitad.

3. La primera línea activa del campo UNO se empieza a barrer en la parte superior izquierda de la pantalla; el campo DOS empieza en el centro de la parte superior.

4. La última línea de barrido activa en el campo UNO termina en el centro de la parte inferior de la pantalla; en el campo DOS termina en la esquina inferior izquierda.

5. En el campo UNO las líneas de información se numeran empezando con el primer pulso igualador; en el campo DOS se numeran empezando con el segundo pulso igualador.

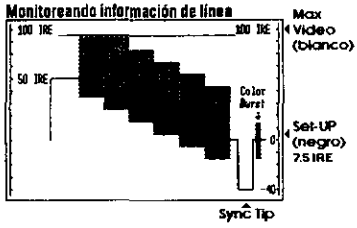
6. En el campo UNO el pulso de sincronía horizontal ocurre media línea después del último pulso postigualador; en el campo DOS el primer pulso de sincronía horizontal ocurre después de una línea completa a partir del último pulso postigualador.

2.2 Grabación y edición

2.2.1 Dispositivos para medición y evaluación de señales de video

Con los métodos clásicos para grabación y edición de video, es necesario cuantificar las componentes de la señal de video desde un punto de vista que no dependa de la percepción individual y por este motivo son utilizados algunos dispositivos que permiten evaluar y corregir las señales de video anteriormente explicadas.

Fig.22 Wave Form Monitor (Monitor de forma de onda)



Esencialmente dos tipos de equipo son necesarios para monitorear y controlar la señal de video básica: el monitor forma de onda y el vectorscopio. El monitor forma de onda mide y despliega una gráfica del nivel de brillo o luminancia y las señales de sincronía. El vectorscopio mide la información de color. En este sentido es posible corregir los errores en la señal debido a malos ajustes en el equipo de grabación antes de comenzar con el trabajo.

Por otra parte al momento de realizar la edición de las partes para obtener la secuencia final se utilizan otros tantos dispositivos, por ejemplo, para que dos segmentos grabados de video se unan en otra grabación como si fueran uno solo es necesario un corrector de tiempo base (TBC).

En este punto cabe hacer notar que ninguno de estos dispositivos están disponibles para realizar este trabajo debido a su alto costo, de cualquier modo, este trabajo será realizado con sistemas de cómputo digitales y las señales de video de imágenes generadas por computadora y hardware especial *generalmente no requieren ningún ajuste extra.*

2.2.2 Tratamiento de la información

Es necesario conocer y tomar en cuenta los alcances y limitaciones inherentes a la realización de una videocinta con fines educativos, para esto se sabe que hay que hacer una selección adecuada y óptima de las imágenes, textos, voz, sonido, música y tiempos de acuerdo a algunos parámetros preestablecidos relacionados con las características del usuario o audiencia final que ayudarán a mantener el interés en la secuencia. En este caso se saben con certeza las necesidades del usuario y la forma de solventarlas, el resto, es más bien un asunto subjetivo o creativo.

De cualquier modo no hay que perder de vista el mensaje que se pretende comunicar con cada artificio técnico, es decir no hay que confundir el medio con el mensaje.

2.2.3 Acceso a la información

La videocinta se debe grabar con los señalamientos y características necesarias o bien acompañarse de formatos externos como por ejemplo: un índice basado en el contador de los reproductores de video para permitir un acceso por parte del usuario final lo más inmediato posible al tema requerido, íconos de identificación en cada tema, etc. En el caso de una posible versión en CD-ROM hay que considerar también un orden lógico para el acceso rápido a los distintos temas, posiblemente mediante menús.

Capítulo 3: *Elección de la tecnología más óptima para realizar el video*

3.1 Máquinas analógicas para video profesional

Como Ingenieros en computación esta opción no sería la más adecuada inclusive desde el punto de vista práctico, pues el proceso es hoy en día muy primitivo considerando el alto costo de estos equipos, llegándose a comparar con su contraparte digital en computadoras modernas adaptadas para este fin.

3.2 Computadoras adecuadas para el trabajo en video con gran capacidad de almacenamiento y proceso (video digital)

Por supuesto que esta sería la opción ideal, existen hoy en día tarjetas para compresión/descompresión MPEG de video en tiempo real que permiten optimizar en gran medida el espacio ocupado en disco duro para guardar una secuencia de gran duración en cuanto a tiempo se refiere. Además se pueden realizar todo tipo de proceso de imágenes con gran facilidad y uno se puede olvidar de los problemas de sincronía en cuanto a imagen, efectos de video, sonido y voz se refiere, todo se realiza de forma digital y se puede retocar una y otra vez sin pérdida de calidad. Pero un equipo con estas características es actualmente muy caro y esta es una razón más que suficiente para considerar que está fuera de alcance. Técnicamente es la solución ideal, sin embargo los costos juegan un papel importante en la toma de decisión.

3.3 Computadoras personales con el auxilio de aparatos de video económicos (video-cámara y/o grabador de videocassette)

Simplemente por el costo es posible deducir que el uso de una computadora personal común es por ahora la solución más adecuada.

Capítulo 4: Selección de la computadora personal con la arquitectura más adecuada para el desarrollo del sistema

4.1 Grabación manual mediante una video-cámara o bien con una tarjeta gráfica de conversión VGA->Video de la pantalla de una computadora compatible con IBM

Es posible realizar y grabar una presentación con ayuda de un programa de autoedición y posteriormente grabar los procesos configuración de forma manual. Definitivamente hacerlo con una videocámara se descarta debido a la pobre calidad. Así que solo resta utilizar una tarjeta de conversión VGA->Video. Pero esto además de ser considerablemente impráctico - si se comete un pequeño error en la grabación hay que hacerlo de nuevo y esto generalmente ocurre si se considera que hay que sincronizar voz, música, audio, movimientos de ratón, etc...- tiene el inconveniente de la baja calidad debido a los problemas de conversión de resolución vertical VGA-NTSC, la necesidad de equipo extra para titulaje, indicaciones, anotaciones y acotaciones sobre el video ya grabado para una para mantener una didáctica adecuada.

4.2 Grabación utilizando programas especialmente diseñados para proceso de video y tarjetas de captura del mismo en computadoras personales comunes (Compatibles con IBM y Macintosh)

Esta debería ser la solución más adecuada en cuanto a computadoras personales se refiere, existen programas para estos fines como el Adobe Premiere, AVID Video Shop, Corel Lumiere, etc. que permiten sincronizar y procesar video y audio de forma totalmente automática y fácil, desafortunadamente la captura de video con este tipo de tarjetas económicas implica una gran pérdida de calidad y por si esto fuera poco la salida en las computadoras personales comunes más potentes (Pentium y PowerPC) es pobre en resolución y cuadros por segundo. Por esta razón definitivamente es casi imposible distinguir las letras pequeñas de las ventanas de WINDOWS, y esto no permite alcanzar el objetivo principal.

4.3 Computadoras personales con tecnologías alternativas

Además de las computadoras personales compatibles con IBM y Macintosh existen muchas otras con características alternativas y precios similares o incluso más bajos. A continuación se listan algunas en orden cronológico y con sus características más importantes. Excepto el BeBOX, todos estos modelos contienen la mayor parte del sistema operativo en ROM. Esto puede ser una gran ventaja cuando el sistema está bien diseñado como se explica a continuación.

4.3.1 AMIGA

Existen muchos modelos: A500, A1200, A2000, A3000, A4000, etc. Todos con capacidades especiales para el trabajo con video y multimedia (gráficos y audio) integradas y sin necesidad de hardware externo. Todos trabajan con un sistema operativo multitarea real escrito en 'C' y ensamblador muy amigable, el sistema apenas ha cambiado en los últimos 12 años, siempre ha sido muy estable y potente (en su mayor parte viene en ROM) así que el software desarrollado es muy extenso. Inclusive en sus últimas versiones (2.0 a 3.1) cuenta con algunas extensiones que permiten soporte multiusuario en red y protección de memoria por hardware al estilo UNIX. Estos modelos cuentan con coprocesadores especializados para el manejo de video en los formatos estándares NTSC y PAL que además permiten una versatilidad gráfica hasta hace poco tiempo imposible en cualquier otra arquitectura similar. Los modelos más pequeños soportan ahora hasta dos procesadores principales, por ejemplo un 68060 trabajando en paralelo con un PPC. El precio del equipo base es mucho menor que el de cualquier MAC PPC o Pentium y las capacidades de expansión son muy grandes.

4.3.2 ATARI

Los modelos más antiguos como el ATARI ST no pueden competir hoy en día, pero los más recientes, conocidos como ATARI FALCON y compatibles tienen una gran ventaja, además de que finalmente soportan multitarea real, todavía con pequeñas limitantes pues es casi imposible para el usuario asignar prioridades para los procesos, incorporan un DSP como estándar a manera de coprocesador que permite auxiliar al procesador principal (generalmente un 68030) en los procesos de imágenes y señales de audio, también es de gran ayuda en procesos de comunicación a alta velocidad. Los modelos ATARI de 16 bits en adelante siempre se han enfocado al desarrollo de audio y música, así que también cuentan con un puerto MIDI como estándar. El problema de este equipo es la falta de software adecuado para aprovechar al máximo las capacidades del DSP. Esto debido al relativamente poco tiempo que ha estado en el mercado, las modificaciones al sistema operativo, el reducido número de usuarios actuales y los problemas de la compañía ATARI misma. Además el precio no es el más adecuado (los modelos avanzados llegan a costar casi tanto como una Pentium) y las capacidades de expansión son algo limitadas.

4.3.3 ACORN

Existen muchos modelos de esta marca, pero su principal característica es que en lugar de inscribirse como todos los demás en los campos de Intel o Motorola, Acorn siempre ha diseñado sus propios procesadores. Por si esto fuera poco, antes de que Motorola comercializara su PPC, Acorn diseñó un procesador RISC (el ARM2) que es tan económico de fabricar como un 68EC020 a 14 Mhz pero con la velocidad de proceso de un 68030 real (alrededor de 8 míps), en estos días todo mundo sabe las ventajas de la tecnología RISC sobre la CISC. Además de esto incluye un sistema multitarea que trabaja muy bien con el procesador RISC -todo en ROM como ya se mencionó; incluyendo librerías, fonts e inclusive algunas aplicaciones para el proceso de textos y un paquete de dibujo-. Las capacidades de expansión son buenas y el precio es intermedio entre los dos modelos anteriores, es decir, más bajo que el de una PC o PowerMAC pero el soporte de software es actualmente bajo sobre todo en las áreas destinadas a video.

4.3.4 BeBOX

Este es un modelo alternativo parecido a una Power Macintosh, su principal ventaja es el sistema operativo pues además de ser multitarea es un sistema abierto y permite una adecuación del mismo a otras arquitecturas con CPU's estándar del mercado de computadoras personales Pentium/PPC. Este sistema operativo en su arquitectura base permite trabajar con varios procesadores PPC (En algunos modelos hasta 7 procesadores) en paralelo y esto es realmente una gran ventaja, por supuesto considerando la relación precio/rendimiento comparando a sistemas similares. Desafortunada y afortunadamente, el resto de la arquitectura se basa en el estándar de expansión PCI del mercado actual de las PC´s, es decir; simplemente programando los manejadores se puede aprovechar casi todo el hardware compatible con las computadoras comunes, pero esto implica que el poder de los procesadores no se podrá aprovechar al máximo, pues el sistema debe hacer una conversión de datos entre el bus de los procesadores y el bus de los dispositivos PCI. En este sentido podemos visualizar que la arquitectura PCI tiene anchos de banda mucho más reducidos, y no podrán ser aprovechados por el proceso en paralelo sobre todo para aplicaciones gráficas de alta resolución en tiempo real. Podemos agregar también que las tarjetas gráficas para la arquitectura PCI utilizan un esquema de memorias separadas para datos y gráficos, por tanto además de todo hay que hacer una transferencia permanente de datos entre ambas memorias para aplicaciones que requieren mucho trabajo con gráficos en tiempo real especialmente video de alta calidad. Todos los problemas anteriores, aunados al costo de una computadora con más de dos procesadores principales PPC y la falta de software nativo para esta arquitectura la presentan como una alternativa interesante, pero hasta cierto punto limitada debido a la falta de software. Esta misma falta de software implica que hay poco conocimiento sobre la estabilidad del S.O. y el hardware. Actualmente el S.O. ha sido compilado para otras arquitecturas basadas principalmente en Pentium's como las PC's y en PPC's como las MAC's pero esto elimina su más grande

ventaja: el soporte de multiprocesadores.

4.3.5 OTROS

Existen una o dos arquitecturas más que se podrían considerar, pero realmente no pueden competir con las anteriores. Además se sabe desde hace tiempo que hay en proceso de desarrollo final otra arquitectura que toma las ventajas de todas las anteriores pero además solventa todos los problemas relacionados con aquellas de forma muy elegante y original superando todas las espectativas del usuario más exigente de computadoras personales. Desafortunadamente hasta el día de hoy el proyecto no se ha finalizado, pero es altamente probable que el próximo año (2000) se termine y comercialice.

4.4 Elección final de la arquitectura

· AMIGA

S.O. estable y robusto desde 1985, muy cercano a UNIX:
(multitarea real, librerías compartibles)
Hardware auxiliar estándar para trabajo con audio/video
Grandes posibilidades de expansión (CPU's, Audio, Video)
Mucho software reciente disponible
Precio muy económico en estos días

· ATARI (Falcon)

S.O. multitarea tipo Windows (1992)
(con todos sus pros y contras)
DSP como coprocesador estándar
Pocas posibilidades de expansión
Poco software nuevo disponible
La compañía ha desaparecido

· ACORN (A3010)

S.O. multitarea real (1992)
(una mezcla de Windows y UNIX)
Primeros CPU's RISC en el mercado de las Computadoras Personales
Pocas posibilidades de expansión
Poco software, sobre todo en el area de video
Precio intermedio entre los dos anteriores.

· BeBOX

S.O. multitarea real (1996)
(tipo UNIX/Linux transportable entre diversas plataformas)
Con el hardware propietario es posible utilizar más de un CPU
Grandes posibilidades de expansión tipo IBM-PC (PCI, etc.)
Poco software disponible
Precio comparable al de una IBM-PC económica.

Haciendo un análisis de las ventajas y desventajas de cada uno de los modelos de computadoras personales se concluye que el modelo AMIGA es el más adecuado para este trabajo.

Capítulo 5: Consideraciones finales sobre la tecnología elegida

5.1 Creación de una secuencia de video digital pregrabado

Para esta plataforma existen desde hace mucho tiempo un sinnúmero de programas para generar o procesar secuencias de imágenes muy sofisticadas en 2 y 3 dimensiones, sin embargo, debido a que estos programas aún no funcionan en tiempo real sin necesidad de hardware adicional, es necesario realizar una pregrabación de las imágenes resultantes para formar una o varias secuencias de video. En esta arquitectura podríamos utilizar los formatos más eficaces para compresión de video y audio: MPEG, MPEG LayerIII, etc. Pero existen otros formatos de video y audio, *algunos exclusivos para esta* plataforma, que economizan los recursos del sistema: Inclusive con un 68030 es posible reproducir video en alta resolución a 60 CPS (entrelazado, "true color") en formatos de video/animación exclusivos de esta plataforma: ANIM8 (8bits), ANIM16, ANIM32, ANIM32 HAM (true color), etc. Aunque no ofrecen una compresión tan alta como la de los formatos MPEG, tienen dos grandes ventajas: no degradan la información durante la edición y ofrecen una gran rapidez de compresión/descompresión. Ahora, si bien estos formatos de reproducción de video permiten una gran versatilidad durante la creación y nos da como resultado un producto listo para su grabación en videocinta, aún existe el inconveniente del requerimiento de espacio y recursos del sistema, además de que en un momento dado será poco práctico trabajar con una cantidad tan grande de información y limitará las posibilidades de retroalimentación con el usuario si se decide la creación de una versión en CDROM.

5.2 Diseño de una estructura multimedia en tiempo real

Es posible realizar un trabajo más adecuado desde el punto de vista de la ingeniería, pues para esta arquitectura también existen muchos programas que permiten generar secuencias de imágenes y efectos de video en tiempo real listos para grabación, que si bien no son tan sofisticados como los pregrabados, permiten una rápida retroalimentación con el usuario y economizan los recursos al máximo, al punto de casi no requerir la intervención del CPU. En este sentido, el uso de video pregrabado será simplemente un complemento adicional que podrá ser utilizado solo cuando sea realmente necesario. Por otra parte, no es necesario grabar toda una pista de audio con una duración de más de una hora y además otra pista para la voz. Si bien es cierto que existen formatos de compresión para audio muy eficaces como el MPEG Layer3, también es cierto que esto volverá el proceso muy lento y problemático en cuanto a la depuración y sincronización se refiere. Por esta razón es conveniente tomar en cuenta otros formatos de audio/música, algunos aún hoy en día exclusivos para esta plataforma, que permiten trabajar de forma más versátil y economizando recursos: MOD, MIDI, S3M, XM, TFMX y etc. Finalmente, lo más importante es realizar una estructura tal que permita al usuario seleccionar lo que realmente desea consultar entre los diversos temas de manera ágil: regresar al tema anterior, ir al menú principal, etc. y de esta forma establecer una mejor retroalimentación con el mismo en los temas clave. Para las necesidades del grabado en video, es posible realizar la grabación de una secuencia de este sistema de forma tal que se toquen los puntos en el orden más adecuado y con otro tipo de ayudas para que el usuario del video pueda acceder de la forma más rápida posible al tema de su preferencia. (Marcas de contador, íconos para cada uno de los temas, etc.)

5.2.1 Uso exclusivo de programas comerciales para esta plataforma para tratar de realizar una estructura lo más adecuada didácticamente hablando para el usuario

Existen diversos programas comerciales y de dominio público que permiten realizar con facilidad una o más facetas de esta estructura, desafortunadamente, una vez estudiados más de 30 programas de este tipo se concluye que ninguno puede funcionar solo; algunos soportan formatos que otros no, pero técnicamente no son lo mejor. Otros trabajan en multiproceso pero no tienen puertos de interconexión para enlaces con el sistema y otros programas. No pocos se apropian de todos los recursos del sistema así que es prácticamente imposible realizar enlaces con otros programas a través del sistema operativo.

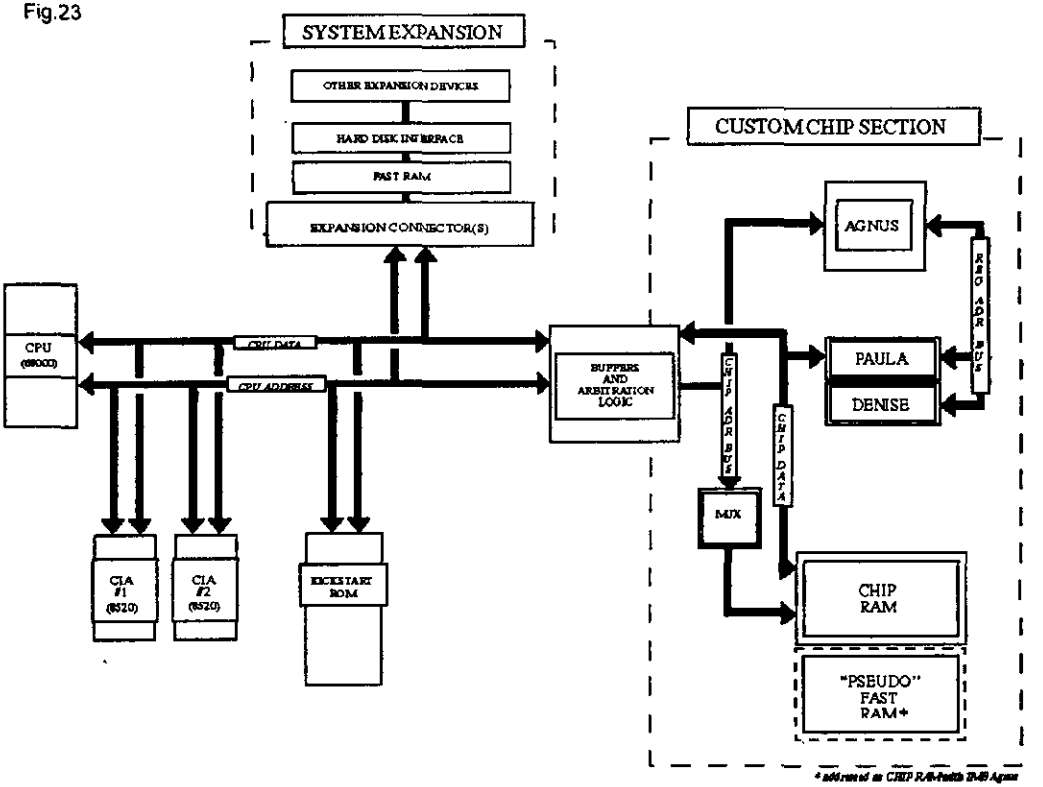
5.2.2 Conocimiento a fondo de la arquitectura para no limitarse a lo que los programas comerciales actuales pueden hacer y realizar enlaces entre nuestro código y el código de estos programas para obtener la estructura final

Desde luego, después de observar el punto anterior podemos entender que esta es la mejor opción desde el punto de vista de la Ingeniería, pues es prácticamente imposible que un programa realice todo lo que queremos sin programar directamente. Para no limitarnos será conveniente conocer la arquitectura a fondo y programarla en el lenguaje más adecuado disponible o bien directamente en ensamblador. Por otra parte, no es necesario reinventar la rueda, si ya existen programas que pueden hacer parte del trabajo hay que aprovecharlos. Existen muchos compiladores para todos los lenguajes como C, C++, Basic estructurado, Pascal, etc. Asimismo existen muchos programas ensambladores que permiten trabajar en bajo nivel y de esta forma se pueden enlazar rutinas especiales para desplegar algunos segmentos de animación o sonido e inclusive modificar y adecuar códigos que se apropian de todos los recursos del sistema operativo y no permiten una comunicación entre programas. Además estos ensambladores permiten trabajar directamente y por lo tanto de la forma más rápida con el hardware mismo del sistema.

Para esta parte es necesario un desarrollo descriptivo que permita explicar la arquitectura a fondo y de esta forma exponer todas sus ventajas. Esto permitirá también tomarla en cuenta como una tecnología alternativa para futuros desarrollos y sobre todo en el campo educativo debido a su versatilidad y bajo costo comparativo con otros sistemas.

5.2.2.1 Un vistazo a la arquitectura

Fig.23



· CPU

Como en todas las arquitecturas computacionales, el CPU es el corazón del sistema. Este es el dispositivo que realiza muchas de las instrucciones relacionadas con cálculos, toma de decisiones y *controla al resto del sistema. El CPU cuenta con los buses tradicionales de acceso a memoria: el bus* de datos y el bus de direccionamiento, ambos de 32 bits, además del bus de control (básicamente usado para especificar lectura/escritura). A diferencia de otras arquitecturas, el bus de datos y el bus de direccionamiento han sido modificados para acceder además a otros dispositivos auxiliares.

Los modelos Amiga más económicos (y en consecuencia los que se usarán en este caso) están basados en la serie de microprocesadores 680x0 (68030 al 68060) Debido al diseño de la arquitectura, en realidad el poder del procesador solo se utilizará para las funciones claves de descompresión de datos en tiempo real, secuencias de control básicas, y el corazón del sistema operativo.

· Conjunto de chips auxiliares (CUSTOM CHIPS)

Algunas actividades consumidoras de tiempo tales como el despliegue de gráficos en movimiento, salidas de audio y acceso a algunos dispositivos de Entrada/Salida como los discos flexibles, se realizan utilizando los coprocesadores y hardware propios para estas funciones. A esta parte de la arquitectura le llamaremos: conjunto de chips auxiliares.

· Memoria RAM y dispositivos DMA

La memoria de esta arquitectura se divide en dos secciones: CHIP RAM y FAST RAM. La memoria CHIP puede ser accedida ya sea por el conjunto de chips auxiliares o por el CPU, debido a esto es necesario el uso de dispositivos de control de actividades de acceso directo a memoria (DMA's), pero esto implica un cierto retraso en el acceso por parte del CPU. Este retraso depende del número de dispositivos que intenten acceder a la memoria CHIP al mismo tiempo, además del hecho de que el conjunto de chips auxiliares tiene prioridad de acceso a la memoria CHIP sobre el CPU.

Por otra parte, la memoria FAST solo puede ser accedida por el CPU, y esto implica que el acceso a esta memoria se realizará a máxima velocidad como en cualquier otra arquitectura, pues el CPU no tiene que competir con el conjunto de chips auxiliares.

Por supuesto el sistema operativo se encarga de realizar la distribución más conveniente almacenando el código de procesador en FAST RAM, si está disponible. Y los gráficos, sonido, buffers de e/s, etc. en CHIP RAM, si el usuario no indica lo contrario.

· Memoria ROM

En esta arquitectura se ha optado por almacenar el corazón de sistema operativo en una memoria permanente, con las ventajas y desventajas ya conocidas. Asumiendo que el sistema es lo suficientemente estable como se ha demostrado, para los modelos más económicos esto implica una mayor velocidad de acceso y una protección extra contra virus y otros problemas. Sin embargo, los sistemas más profesionales cuentan con MMU's o unidades de manejo de memoria que permiten una protección casi infalible del código en RAM por hardware y de este modo los ROM's se hacen casi innecesarios.

· Dispositivos CIA

Los CIA's (*Complex Interface Adaptor*) manejan el flujo de datos de los puertos de E/S, el CPU está conectado a estos, así que puede saber en todo momento el estado de los puertos mediante los registros del CIA, como son los registros de control (para configuración de los puertos), los registros de datos (para enviar y recibir datos) y algunos registros especiales que informan al CPU sobre el estado de los buffers de memoria de los puertos (lleno, vacío, etc.), permiten conteos de tiempo y activan interrupciones relacionadas con estos eventos.

5.2.2.2 Conociendo el procesador central (CPU)

· Lenguaje ensamblador

Para ejecutar una instrucción en memoria, el CPU debe decodificar y ejecutar la instrucción en cuestión, la parte relacionada con la decodificación es responsable de enviar las señales de control apropiadas para realizar la operación requerida, por ejemplo una transferencia de datos, una operación lógica, etc. Todo esto al momento de ejecutar una instrucción. Sin embargo, si la serie de operaciones es muy compleja para una instrucción específica, entonces el decodificador dentro del CPU usualmente implementa una técnica de microcodificación que permite ejecutar una serie programada de acciones para cada instrucción. Este tipo de CPU's son conocidos como CISC, aquellos que no emplean microcodificación son llamados CPU's RISC.

No es necesario programar con soltura directamente con las instrucciones propias del microprocesador, pues existen lenguajes estándares como C/C++ e inclusive una especie de lenguaje BASIC muy poderoso con características de programación propias de cualquier compilador de C que permite obtener un código de salida para muchas funciones casi tan óptimo como el mismo compilador de C. Además

algunos modelos de esta arquitectura utilizan uno o más microprocesadores RISC PowerPC en paralelo con un CISC de la familia 680x0 y esto implica que es mejor dejar que el compilador y el sistema operativo se encarguen de la generación y ejecución del código, pues realizar un trabajo óptimo con más de un procesador en bajo nivel no es algo muy sencillo.

Es importante mencionar que si se decide utilizar la programación en ensamblador para optimizar rutinas clave no hay que realizar código automodificable para así poder aprovechar la velocidad de la memoria cache con la que cuentan los CPU's modernos.

Consultar el Apéndice 3 para una descripción más detallada sobre el lenguaje ensamblador del 68000.

5.2.2.3 Acceso al hardware: gráficos, audio, coprocesadores, etc.

· Acceso a los registros del conjunto de chips auxiliares.

Los chips para el manejo de gráficos, sonido y e/s requieren datos para trabajar y también permiten que otros sean leídos, de cualquier modo, estos chips requieren de alguna clase de almacenes para mantener sus parámetros, estos almacenes son llamados registros, y son de 16 bits cada uno.

Estos registros son mapeados en memoria, esto significa que el CPU ve estos registros de la misma forma que ve la memoria RAM. El 68000 lee y escribe en esos registros exactamente de la misma forma que al escribir en RAM. Los registros comienzan en la dirección $dff000 y terminan en $dffffff, de este modo al escribir o leer en alguna localidad entre estas direcciones, en realidad se están utilizando los registros del conjunto de chips auxiliares y no la memoria.

· Mapas de Bits o despliegues gráficos

En las tarjetas de despliegue gráfico modernas, utilizadas en todas las arquitecturas, los mapas de bits se componen de pixels, donde cada pixel corresponde a una localidad de la memoria gráfica que puede ser de 8,16 o 32 bits. Es bien sabido que un máximo de 24 bits permite un despliegue de todos los colores visibles por el ser humano, pero los 8 bits extra son utilizados en las tarjetas más avanzadas para efectos de canal alfa, como transparencias, etc. En la arquitectura AMIGA es posible instalar alguna tarjeta moderna de 32 o inclusive 64 bits para alcanzar muy altas resoluciones, pero esto implica un cierto costo extra que para nuestra aplicación no es necesario, pues estas altas resoluciones no son necesarias para trabajo en video, así que lo mas conveniente por ahora es utilizar únicamente el conjunto de chips auxiliares.

· Planos de bits para formar despliegues gráficos

El hardware gráfico del conjunto de chips auxiliares ofrece una forma de despliegue distinta a las tarjetas gráficas comunes. En lugar de requerir los 8, 16 o 24 bits por pixel, permite generar una pantalla completa con un solo bit de información por pixel o un plano de bits. Esto significa que los despliegues gráficos se conforman de planos de bits, de este modo es posible ahorrar mucha memoria, pues solamente se utilizaran el número de planos requeridos para el número de colores deseados de acuerdo a la siguiente fórmula: número de colores=2^número de planos

Además de esto hay que agregar que el hardware gráfico permite recomponer el despliegue en cualquier línea de la pantalla, cualquier número de veces, esto implica que de la línea 1 a la 16 podemos tener un despliegue con 2 colores, de la línea 17 a la 64 un despliegue de 256 colores, etc. Inclusive es posible mover uno o más mapas de bits con independencia de los otros por la pantalla o el segmento de pantalla deseado, esto permite algunos efectos gráficos en tiempo real que solamente arquitecturas muy avanzadas y poco económicas podrían realizar.

Por otro lado este formato de despliegue tiene algunos inconvenientes, por ejemplo, podemos pensar en

un programa de polígonos mapeados en 3D, en tiempo real, en donde es necesario cambiar cada pixel del despliegue de la forma más rápida posible. Con una tarjeta gráfica moderna común, lo anterior implicaría una sola instrucción del CPU cambiando la localidad de memoria en donde se encuentra el pixel en cuestión, pero con el formato de despliegue de los chips auxiliares esto implica leer, hacer una o dos operaciones extra y escribir en cada uno de los planos que conforman la imagen, lo cual aumenta considerablemente el tiempo requerido para desplegar un solo pixel.

Para solventar esto podemos utilizar el coprocesador auxiliar llamado Blitter, que programado adecuadamente puede realizar esta conversión en tiempo real y con independencia del CPU, que únicamente requiere cambiar una sola localidad de memoria tal y como si estuviera trabajando con una tarjeta gráfica real. El problema aquí surge cuando se requiere una pantalla de muy alta resolución, pues el coprocesador auxiliar no tiene la velocidad que se requiere para ello y se hace necesaria la ayuda del CPU o bien la instalación de la tarjeta gráfica misma. Sin embargo, para despliegues a resoluciones compatibles con video NTSC no existe problema.

· Generación de la señal de video para el despliegue en pantalla

Como ya se mencionó, esta arquitectura puede desplegar porciones de mapas de bits mucho más grandes que la pantalla misma, con movimiento fino en todas direcciones. Además puede desplegar y controlar cada plano del mapa de bits separadamente, de tal forma que se puede formar por ejemplo, un despliegue dual, en donde existan dos mapas de bits, cada uno formado por la mitad del total de planos que se utilizarían para una sola imagen.

Adicionalmente, el hardware gráfico permite desplegar 8 imágenes extra, llamadas *sprites*, de anchura menor que un mapa de bits pero con una gran flexibilidad en cuanto a posición y movimiento en la pantalla. De cualquier modo, alineando los *sprites* adecuadamente es posible realizar virtualmente un mapa de bits extra, totalmente independiente de los mapas de bits formados por planos.

Desafortunadamente no está disponible para los usuarios la información oficial de todos los registros que controlan las señales de video, pero de cualquier modo el sistema operativo ya contiene muchos valores y configuraciones predefinidas que permiten obtener señales de video con resoluciones horizontales en pixels que van de 320 hasta 1280+overscan. Por otra parte, las resoluciones verticales van de 200 hasta 1024+overscan.

Desde luego, algunas de estas resoluciones solamente son posibles con un monitor multisincrónico, otras con monitores tipo VGA, pero las resoluciones compatibles con video NTSC son más que adecuadas, podemos tener por ejemplo una pantalla de 1280x400 que puede ser convertida por el hardware en una señal NTSC, que si bien es casi imposible que conserve todo el detalle de resolución horizontal, ofrece una mejor calidad que cualquier otra resolución horizontal común como 640 o 320 pixels.

Unicamente podemos mencionar que esta arquitectura utiliza aproximadamente 10 registros conocidos del conjunto de chips auxiliares para formar y describir la señal de video, que será enviada ya sea a una pantalla NTSC/TV, un monitor VGA, o un monitor multiscan/multisincrónico. Estos registros son: HBSTRT, HSSTRT, HSSTOP, HBSTOP, TOTCLKS, VBSTRT, VSSTRT, VSSTOP, VBSTOP, TOTROWS.

Sabiendo las características generales de la señal de video podemos deducir las correspondencias:

HBSTRT/HBSTOP=inicio/fin del intervalo de borrado horizontal.
HSSTRT/HSSTOP=inicio/fin de la señal de sincronía horizontal.
TOTCLKS/TOTROWS=frec. de barrido horizontal y frec. de barrido vertical.
VBSTRT/VBSTOP=inicio/fin del intervalo de borrado vertical.
VSSTRT/VSSTOP=inicio/fin de la señal de sincronía vertical.

El rango de valores y su significado son desconocidos para nosotros, de cualquier modo, como se menciona anteriormente, es posible experimentar con ellos en un monitor multisincrónico para observar sus efectos y tratar de obtener otras señales de video RGB por si se desea compatibilidad con algún monitor no estándar o inclusive futuro, simplemente cabe hacer la advertencia, esta sí mencionada en la documentación, de que la modificación de algunos de estos registros sin cuidado puede llegar a dañar o incluso destruir algunos monitores.

Solo resta mencionar de nuevo que las resoluciones predefinidas en el sistema operativo permiten utilizar virtualmente cualquier dispositivo de despliegue de video, así que en general no es necesario tocar estos registros. En nuestro caso, por ejemplo, solamente nos interesa la señal NTSC y esta es generada perfectamente por el sistema operativo.

· Registros de posición y dimensión del área visible.

Una vez que se tiene la salida correcta de video NTSC, existen otros registros que permiten posicionar y cambiar de tamaño el área de despliegue, estos registros, llamados: DIWSTRT, DIWSTOP, DDFSTRT y DDFSTOP indican al hardware gráfico en que momento comenzar a desplegar mientras el haz de electrones explora la pantalla. También indican cuando detener el despliegue y que longitud debe tener. Todo ello para definir un área cuadrada de despliegue.

Afortunadamente, el sistema operativo también tiene funciones que predefinen y calculan los valores para cualquier tamaño de pantalla requerido, así que cualquier lenguaje de programación simplifica esto. No es necesario conocer y programar directamente estos registros para ganar velocidad o versatilidad, pues el área de despliegue generalmente permanece estática por lo menos durante la ejecución y uso de un programa. En nuestro caso, el área de despliegue NTSC también permanecerá constante.

· Registros de generación y desplazamiento de mapas de bits.

Estos registros permiten especificar las posiciones de memoria en donde se encontrarán los planos de bits que forman el mapa de bits a desplegar, en realidad, puesto que cada registro es de 16 bits, son necesarios dos registros BPLxPTH y BPLxPTL (x=plano de bits) para especificar una dirección de memoria en formato de 32 bits, esta dirección debe estar en la memoria CHIP.

Como se mencionó anteriormente, es posible crear un mapa de bits de un tamaño mucho mayor al tamaño de la pantalla de visualización real del monitor, a este mapa gigante de bits le podemos llamar pantalla virtual, en este sentido, es necesario ignorar los datos de la pantalla virtual que no deben aparecer en la pantalla real durante el despliegue, para esto existen dos registros adicionales: BPLxMOD (x=1 planos impares,x=2 planos pares) que indican el número de bytes a saltar al final de cada línea del despliegue, este número de bytes es llamado módulo. Por ejemplo, si el despliegue tiene 320 pixels de ancho (40 bytes) y el mapa de bits o pantalla virtual tiene 480 pixels de ancho (60 bytes) entonces el módulo tiene que ser de 20 bytes.

El valor del módulo, correctamente utilizado, permite además realizar algunos trucos como por ejemplo desplegar el mapa de bits invertido en Y simplemente inicializando los registros apuntadores de planos de bits (BPLxPT) al comienzo de la última línea del mapa de bits y dándole un valor negativo a los registros del módulo para mover los apuntadores hacia atrás dos líneas después del final de la línea actual para comenzar en la anterior.

Es claro que para desplazar un mapa de bits verticalmente, basta con modificar el valor de los registros apuntadores de planos de bits sumándoles -movimiento hacia abajo- o restándoles -movimiento hacia arriba- el valor del ancho del mapa de bits.

El desplazamiento horizontal simplemente se realiza sumando o restando un determinado número de

palabras (16, 32 o 64 bits dependiendo del modo gráfico y la arquitectura), pero esto no permite un desplazamiento fino, pixel por pixel. Para solventar esto se utiliza un registro extra: BPLCON1 que permite desplazar el mapa de bits horizontalmente de 1 a 16 pixels (1 a 32 ó 1 a 64 pixels dependiendo del modo gráfico y la arquitectura). Afortunadamente, el sistema operativo y todos los lenguajes para esta arquitectura realizan esto de forma automática mediante funciones especiales altamente optimizadas que toman en cuenta el modo gráfico y la arquitectura en cuestión, así que esto es transparente para el programador.

· Copper (Coprocesador)

El Copper es el coprocesador más simple de esta arquitectura, pero es muy poderoso en el sentido de que está estrechamente ligado con el hardware de despliegue. Tiene tres funciones o comandos básicos, WAIT - permite esperar una determinada posición del haz rastreador que genera las líneas de video, MOVE - permite modificar el valor de algún registro del conjunto de chips auxiliares, SKIP - permite saltar o ignorar la siguiente instrucción MOVE si el haz rastreador se encuentra en alguna posición determinada.

Por ejemplo, podemos escribir un programa en pseudocódigo como el siguiente:

```
WAIT for line 0 pixel 0
MOVE $000 to colour reg 0
WAIT for line 10 pixel 0
MOVE $111 to colour reg 0
WAIT for line 20 pixel 0
MOVE $222 to colour reg 0
: etc.
```

que resultaría en bandas horizontales sobre el color de fondo (registro de color 0).

El código de arriba no funciona en la vida real, pues un programa para el Copper se compone de series de números de 16 bits, pero como antes se mencionó, el sistema operativo facilita el trabajo pues cuenta con funciones que permiten realizar esto en un formato parecido al pseudocódigo.

En este sentido es posible realizar un programa para este coprocesador que se ocupe por completo del despliegue en pantalla modificando los registros adecuados en el momento correcto. Y de este modo el o los CPU's y otros coprocesadores se pueden ocupar por completo de otras tareas para las que son más eficaces.

· Blitter (Coprocesador)

El Blitter es un coprocesador mucho más complejo que el Copper, su función principal es copiar segmentos de un mapa de bits, realizar cálculos lógicos, líneas y otras funciones gráficas en 2D consumidoras de tiempo. Esto libera al CPU de realizar funciones simples pero repetitivas como el movimiento de una ventana por la pantalla -que en otras arquitecturas llega a consumir más del 50% del CPU por algunos momentos-. De este modo el CPU y el coprocesador matemático se pueden dedicar casi al 100% a las funciones de cálculo, control e inteligencia artificial para las que fueron realmente diseñados.

Una vez más, no es necesario comprender a fondo el funcionamiento y programación de este coprocesador, pues el sistema operativo contiene las funciones esenciales para hacerlo funcionar cuando y como se indique, en paralelo con el CPU.

La referencia técnica de los principales registros de hardware se encuentra en el Apéndice 2.

5.2.2.4 Introducción al Sistema Operativo

El sistema operativo de esta arquitectura, desde siempre ha sido multitarea real y esto obliga a que todos los programas residentes en memoria puedan compartir los recursos del sistema de la forma más óptima posible, por esta razón, los compiladores permiten generar códigos que únicamente requieren estar una vez en memoria para poder ser ejecutados cualquier número de veces, simplemente requiriendo memoria para las variables por cada ejecución. De este modo es posible generar librerías y programas compartibles.

· Librerías compartidas

Como se ha mencionado anteriormente, el sistema operativo cuenta con muchas funciones para el manejo del hardware y los recursos de la arquitectura. Estas funciones se encuentran en una serie de librerías compartidas (*shared libraries*) que permiten su uso por parte de muchos programas al mismo tiempo, sin necesidad de multiplicar el código principal de las mismas, esto significa como ya se mencionó, que el código de una librería, solamente tendrá que estar una vez en memoria para ser usado por cualquier número de programas que así lo requieran.

Para acceder a las funciones de estas librerías, simplemente es necesario "abrir" la librería correcta llamando a la función: OpenLibrary() que regresa un apuntador a la estructura base global de la librería requerida. La función OpenLibrary() se encuentra en el corazón o la librería principal del sistema (EXEC), que por supuesto, siempre está lista para usarse en memoria.

Las librerías principales se encuentran en ROM, y las librerías complementarias se encuentran en el disco del sistema. A continuación se describen las más importantes.

· Librerías principales (ROM)

Estas librerías siempre están disponibles, pero necesitan ser abiertas antes para poder usar sus funciones. Aquí esta la lista de las más importantes en ROM.

EXEC

Esta es la librería más importante pues controla la memoria, los procesos, librerías, periféricos, recursos, puertos de E/S, etc. Es responsable del correcto funcionamiento del entorno mutiproceso.

DOS

Esta librería, también llamada Amiga D.O.S. contiene las rutinas que abren, cierran, leen y escriben archivos y todo lo relacionado con estos.

GRAPHICS

Contiene todas las funciones que tienen que ver con gráficos, como son los despliegues de mapas de bits, dibujo y animación en pantalla, *sprites*, texto, fuentes, etc.

LAYERS

Contiene rutinas que manejan la pantalla de tal forma que diferentes capas de despliegues gráficos pueden ser usados sin problema aunque se encimen uno con otro, por ejemplo las ventanas del sistema.

INTUITION

Se ocupa de la interfaz de usuario, como son las pantallas, ventanas, botones, menús, etc. y trabaja conjuntamente con las librerías: *Graphics* y *Layers*.

MATHFFP

Contiene rutinas optimizadas para operaciones matemáticas básicas en punto flotante de precisión simple.

RAM-LIB

Maneja el disco virtual que se puede crear en RAM.

EXPANSION

Maneja los puertos de expansión.

· Librerías complementarias (disco)

Si un programa abre una librería que se encuentra en disco, las rutinas *AmigaDOS* buscarán en el periférico lógico llamado LIBS:, que ha sido asignado a el directorio en disco 'libs' del sistema, y entonces cargará la librería completa en RAM. Si la librería ha sido ya cargada en RAM por otro proceso, las rutinas *Exec* usarán entonces la librería ya existente. A continuación se listan las librerías más importantes de este tipo.

TRANSLATOR

Esta librería convierte cadenas del lenguaje inglés en cadenas fonéticas que pueden ser escuchadas por la salida de audio.

DISKFONT

Despliega y maneja las fuentes existentes en el disco.

ICON

Despliega y maneja los iconos asociados con los archivos de disco usados en el ambiente gráfico llamado *Workbench*.

MATHTRANS

Contiene las funciones: sen, cos, log, ln, raíces, potencias, etc.

MATHIEEEDOUBBAS

Esta librería es casi idéntica a la librería "MATHFFP" en ROM, pero usa precisión doble.

· Uso de las librerías compartidas

Como se mencionó anteriormente, las librerías se abren llamando a la función OpenLibrary(). Esta regresará un puntero a una estructura de la misma librería, y este puntero debe ser almacenado en una variable puntero con un nombre especial especificado en la documentación oficial. Por ejemplo, si se abre la librería *Intuition*, el apuntador debe llamarse "IntuitionBase", y si se abre la librería *Graphics*, el apuntador debe llamarse "GfxBase".

Sintaxis: pointer = OpenLibrary(name, version);

pointer: (struct Library *) Apuntador a la estructura base de la librería en cuestión, esta estructura base tiene un nombre especial que generalmente inicia con el nombre de la librería y termina con el sufijo Base: "IntuitionBase", "GfxBase", etc. Referirse a la documentación oficial para saber los nombres exactos para cada librería.

name: (char *) Apuntador a una cadena conteniendo el nombre de la librería, por ejemplo la librería *Intuition* es llamada "intuition.library", la librería *Graphics* es llamada "graphics.library", etc.

version: (long) Aquí se especifica la versión mínima de la librería requerida para el programa en cuestión. Para usar cualquier versión simplemente se especifica un 0.

Cada librería que ha sido abierta debe cerrarse antes de que el programa termine, esto se realiza llamando a la función CloseLibrary().

Sintaxis: CloseLibrary(pointer);

Pointer: (struct Library *) Apuntador a la estructura base de la librería que ha sido previamente inicializada por una llamada a OpenLibrary().

Aquí se muestra un fragmento de un programa que abre y cierra la librería *Intuition*. Cabe hacer notar el uso de un operador *cast* o molde para forzar a que el valor regresado por la función efectivamente apunte a la estructura base de la librería.

```
struct IntuitionBase *IntuitionBase;

main()
{
  /* Open the Intuition Library (any version): */
  IntuitionBase = (struct IntuitionBase *) /* cast */
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */
  ... ...

  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );
}
```

A continuación se muestra otro fragmento, que esta vez abre y cierra la librería Graphics:

```
struct GfxBase *GfxBase;

main()
{
  /* Open the Graphics Library (any version): */
  GfxBase = (struct GfxBase *) /* cast */
    OpenLibrary( "graphics.library", 0 );

  if( GfxBase == NULL )
    exit(); /* Could NOT open the Graphics Library! */
```

```
  /* Close the Graphics Library: */
  CloseLibrary( GfxBase );
}
```

De este modo, un programa en cualquier lenguaje puede tener acceso a todas las funciones del sistema operativo que simplifican mucho trabajo, sobre todo el hecho de trabajar con más de un microprocesador a la vez. Es importante recordar que el sistema se basa en el lenguaje C, así que la mayoría de las funciones implicarán el uso de estructuras y otros elementos del mismo lenguaje, que son también accesibles desde lenguaje ensamblador si es necesario.

En el Apéndice 1 se muestra una lista de las funciones más importantes del sistema operativo de esta arquitectura.

5.2.2.5 Programación en ambientes multitarea o multiusuario

Independientemente de la arquitectura, es necesario tomar en consideración algunas recomendaciones importantes para el trabajo correcto en un entorno multitarea, multiproceso ó multiusuario.

· Conocimiento de la arquitectura

Si la arquitectura utiliza más de un microprocesador, generalmente el sistema operativo se encarga de dividir el trabajo de la forma más óptima entre estos, pero es un hecho que el tener una idea de lo que la máquina realiza internamente permite, en determinadas situaciones, una asignación "manual" de tareas y recursos de hardware mucho más óptima que la que cualquier sistema operativo podría realizar.

· Optimización en el uso del CPU

Los sistemas multitarea ejecutan muchos programas "simultáneamente" dividiendo el tiempo del procesador entre todos ellos de acuerdo a las prioridades asignadas por el sistema operativo o el usuario mismo, de cualquier modo, comúnmente el usuario no asigna prioridades a cada programa así que esta responsabilidad generalmente recae en el sistema operativo.

Pero el sistema no siempre asigna el tiempo de procesador óptimo al programa que así lo requiere, así que el programador debe tratar de realizar un código lo suficientemente adecuado como para liberar todo el CPU posible siempre que sea necesario. Por esto, siempre que se espera la ocurrencia de un evento externo utilizando un bucle repetitivo, se recomienda utilizar dentro del bucle las funciones de espera implementadas en el sistema operativo como son: delay(), sleep() o wait() con un valor de tiempo adecuado al evento en cuestión (1/60s en adelante). O en su defecto utilizar la función del sistema para reconocer el evento correspondiente si es que esta existe. Esto evitará que el CPU se sobrecargue ejecutando un bucle de instrucciones repetitivas miles de veces por segundo cuando podría utilizar ese tiempo para códigos más importantes.

· Comunicación correcta entre procesos

Si el sistema multitarea permite desactivar los esquemas de protección de memoria, es posible establecer una comunicación entre procesos de manera primitiva, simplemente asignando a los procesos en cuestión una dirección de memoria común que todos pueden leer y modificar. Sin embargo, el sistema operativo de la arquitectura AMIGA y algunos otros implementan puertos de comunicación que pueden ser asignados a cada uno de los procesos creados, de este modo, es posible establecer la comunicación entre ellos de una forma más elegante y controlada, inclusive si los esquemas de protección de memoria mediante software o hardware no pueden ser desactivados para los procesos en cuestión.

· Uso adecuado de recursos no compartibles

Es importante recordar que cuando un proceso o programa requiere utilizar un recurso o periférico compartible del sistema, como puede ser un archivo abierto para escritura, los canales de audio, o el puerto paralelo, es necesario bloquear el recurso para otros procesos si se encuentra libre y liberarlo lo más pronto posible para que otro proceso pueda utilizarlo. Desde luego esto se debe realizar con las funciones apropiadas del sistema operativo para evitar conflictos.

· Manejo de interrupciones

Usualmente, una interrupción es una señal que viene de un dispositivo externo al CPU. Una vez que esta señal es recibida, el CPU salva los registros necesarios y salta a una subrutina especial llamada la rutina de servicio de interrupción (ISR). La ejecución y localización de esta rutina está determinada por el tipo de interrupción y el vector de interrupción (dirección en memoria de la ISR) entre otras cosas. Después de que esta rutina es ejecutada, el CPU restaura todos los registros necesarios y continua desde donde se detuvo como si nada hubiera sucedido.

El CPU maneja algunos niveles básicos de prioridad para cada tipo de interrupción, pero es el sistema operativo el que se encarga de asignarlos de acuerdo al hardware y el software del sistema. Aunque los sistemas operativos en general se encargan de controlar y dar servicio a las interrupciones más importantes o comúnmente utilizadas. En la arquitectura AMIGA, es posible añadir o reescribir las rutinas de servicio a interrupciones por hardware o software de manera simple y perfectamente compatible con el entorno multitarea. Esto puede ser muy útil en aplicaciones personalizadas que requieren una respuesta inmediata en tiempo real a las señales del hardware o dispositivo externo, sin importar si la aplicación en cuestión posee o no el tiempo o la prioridad del CPU en el momento clave.

Capitulo 6: Desarrollo e implementación del sistema

6.1 Desarrollo de un guión indicando puntos principales y tiempos aproximados de duración

Una vez elegido el camino, nos enfocamos al desarrollo del sistema propiamente dicho, lo principal es explicar de forma clara temas como el de la introducción a Internet, la configuración del software de conexión y el uso de los programas clientes. Por este motivo se realizaron los siguientes guiones para cada tema, que posteriormente serán grabados con voz, sincronizados y mezclados con el video o sistema multimedia final.

· Introducción a Internet:

La comunicación, entendida como el acto de compartir información, es una de las actividades sustanciales de los seres humanos. Cuando las posibilidades de comunicación inherentes al ser humano fueron insuficientes, éste crea un sinnúmero de sistemas y artefactos para expresarse: escritura, radio y televisión son los más revolucionarios. Todos, en especial la televisión, permiten reflejar con claridad lo que uno piensa o siente, pero también tienen una limitante, no permiten una retroalimentación emisor - receptor y su acceso es restringido en mayor o menor medida. Es aquí donde entran en auxilio el correo y el teléfono, los dos permiten establecer una comunicación en ambos sentidos y complementan a los anteriores.

Si entendemos el reflejo del pensamiento humano como información, podemos comprender la urgente necesidad de tratar o procesar esta misma información pero con mayor rapidez y efectividad, es en este momento cuando se piensa en desarrollar la primer computadora.

El desarrollo de la computación permitió procesar información a gran escala, a tal grado que hoy en día casi todas las actividades importantes de la vida moderna dependen del funcionamiento de al menos una computadora.

De este modo, para optimizar el proceso de los grandes niveles de información que se manejaron en una sola computadora, se consideró otra vez la misma fórmula: comunicación, esta vez aplicada a las máquinas, a partir de este momento se creó la primera red de computadoras que permitió descentralizar y compartir la información.

Una vez que las telecomunicaciones se desarrollaron y los enlaces entre computadoras crecieron tanto como los enlaces telefónicos en todo el mundo, se encontraron nuevas y revolucionarias aplicaciones para esta red, cada computadora conectada puede servir como un instrumento de comunicación lo bastante flexible y poderoso como para satisfacer las necesidades de cualquier ser humano en busca de información sin las restricciones de cualquier otro medio de comunicación.

En Internet podemos encontrar información sobre cualquier tema, esta información puede estar en forma de hojas de texto con fotografías, sonidos o incluso video. También se pueden enviar mensajes electrónicos que llegan en segundos y pagar algún producto utilizando los números de su tarjeta de crédito.

Pero sobre todo uno puede compartir su propia información con toda la gente que así lo desee y además comunicarse en cualquier momento con cualquier persona conectada en cualquier otra parte del mundo ya sea mediante texto, utilizando la propia voz en conexiones rápidas, o incluso con una cámara de video conectada a la computadora.

Para hablar de Internet como se conoce actualmente, es conveniente entender algunos conceptos básicos relacionados con la comunicación en una Red de computadoras. Primero cabe hacer notar que para compartir información, es decir para que exista comunicación es necesario contar con tres elementos: emisor, receptor y medio de transmisión.

Es importante mencionar que para lograr la comunicación el emisor y el receptor deben utilizar un lenguaje común.

El problema del medio de transmisión está resuelto parcialmente, desde tiempo atrás han existido los medios: como los cables y las ondas electromagnéticas, pero algunos aún no tienen la velocidad o el alcance necesarios. Por otra parte el problema relacionado con el lenguaje común se resolvió en un principio utilizando algunas reglas básicas de comunicación para computadoras.

Estas reglas, después conocidas como protocolos de comunicaciones, eran muy simples y dependían de la arquitectura utilizada, por esta razón solamente computadoras similares podían ser conectadas.

La mayoría de las redes en Internet se componen de dos tipos de máquinas: servidores, también conocidos como anfitriones o hosts y clientes, el servidor proporciona información o servicios que el cliente puede acceder en determinados niveles. Es posible que un servidor funcione también como cliente.

Bajo estas condiciones surgieron las primeras redes, privadas y usualmente restringidas a pequeñas regiones. El primer proyecto realmente ambicioso que se puede considerar como el principal antecedente de Internet se llamó ARPANET, una red creada en los E.U.A. alrededor de 1969.

Conforme se desarrollaron los medios de comunicación para largas distancias, algunas redes establecieron conexiones con redes de otras partes del mundo y en determinado momento ya se podía hablar de una gran red con cobertura internacional.

Para entonces los protocolos de comunicación utilizados dependían en gran medida de la arquitectura propia de cada red y en realidad las funciones que estos proporcionaban eran prácticamente las mismas, esto se convirtió en un gran obstáculo para una comunicación óptima. Pero a partir de 1983 se hace un cambio generalizado hacia un protocolo de comunicaciones llamado TCP/IP: Transport Control Protocol/Internet Protocol por sus siglas en inglés o bien: Protocolo de Control de Transporte/ Protocolo de Internet, esto permitió que redes y computadoras de arquitecturas totalmente distintas pudieran comunicarse entre sí simplemente siguiendo el nuevo protocolo, de esta forma se resolvió el problema del lenguaje común necesario para la comunicación.

Posteriormente se permitió el acceso para casi cualquier red institucional, y es entonces cuando se vislumbran los inicios de la red Internet actual.

· Servicios que ofrece RedUNAM/Internet:

Hoy en día Internet es una red global de computadoras comunicadas entre sí, Esta red global está formada por redes más pequeñas o subredes regionales. La RED INTEGRAL DE TELECOMUNICACIONES DE LA UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO: R.I.T.UNAM, proporciona a RedUNAM todos los servicios relacionados con Internet y paralelamente brinda servicios de voz, datos y video a las dependencias universitarias ubicadas en el D.F., área metropolitana y el interior de la República. R.I.T.UNAM representa una poderosa herramienta para acercar a los estudiantes, académicos e investigadores a muchísima información reciente y a las más avanzadas tecnologías de comunicación integral.

Internet permite una comunicación global entre las computadoras conectadas, por lo tanto también puede haber una comunicación entre usuarios. Esta comunicación se logra mediante los diferentes servicios que soporta RedUNAM/Internet: Correo Electrónico, FTP, WWW, Servidor de Nombres, y otros.

Para enviar y recibir mensajes a través de Internet hay que obtener una cuenta de correo electrónico; esta cuenta hará el papel de un apartado postal localizado dentro de una oficina de correos, en términos de Internet, esta oficina se conoce como: servidor, host o anfitrión.

La cuenta o dirección de correo se identifica con 2 datos importantes: la clave de usuario y el nombre del servidor: clave_del_usuario@nombre_del_servidor

El servidor de correo puede estar en cualquier parte del mundo y generalmente estará funcionando las 24 horas del día durante los 365 días del año, esto permite utilizar los servicios para enviar y recibir correspondencia en cualquier momento y desde cualquier otra parte del mundo que cuente con una conexión a Internet.

La correspondencia será almacenada en el espacio que hace el papel de apartado postal, para hacer uso de este, es decir, para ver la correspondencia se asigna una llave conocida como password de correo. Gracias a esta llave podemos mantener la privacía de los mensajes.

En RedUNAM existen muchos servidores de correo electrónico, el servidor central lleva por nombre servidor.unam.mx, así que la dirección electrónica de un usuario de este servidor es clave_del_usuario@servidor.unam.mx

FTP es un programa que permite realizar una transferencia de archivos en Internet. Mediante un programa cliente podemos conectarnos a alguna computadora que cuente con este servicio y obtener ó enviar archivos de todo tipo como programas, imágenes y textos.

El WWW es hoy en día el método más común para presentar información en Internet, su interface permite que el usuario tenga acceso a muchísima información de forma simple y amigable. Actualmente RedUNAM/Internet cuenta con servidores que prestan este servicio ofreciendo información multimedia.

Para comunicarnos con otra persona por teléfono es necesario conocer su número telefónico, si consideramos que los teléfonos también conforman una red, podremos entender entonces que en una red de computadoras cada una de estas necesita un identificador.

Cada computadora conectada a Internet se identifica con una direccción IP, esta dirección se compone de cifras numéricas. Para hacer más fácil la localización y memorización de estas direcciones existe un servicio llamado DNS, un servidor del mismo toma la dirección numérica y hace una correspondencia con palabras separadas por puntos.

La primera palabra nos indica el nombre de la máquina o bien el tipo de servicio que presta.

Posteriormente aparece el dominio que indica la organización y ubicación geográfica a la que pertenece la máquina, este puede estar conformado por una, dos o más palabras.

Algunas direcciones de correo se identifican únicamente con el dominio y por supuesto la clave del usuario.

La última palabra del dominio es una abreviatura que indica el país en donde se encuentra físicamente la computadora excepto máquinas pertenecientes a los E.U.A. y algunos otros casos especiales.

Para el soporte de todos los servicios de información relacionados con Internet y RedUNAM misma, la Dirección de Telecomunicaciones tiene a su cargo la Subdirección de Redes y la Coordinación de Conectividad.

La Subdirección de Redes se ocupa de todo lo relacionado con el correcto funcionamiento de la parte lógica de RedUNAM, Para ello cuenta con lo siguiente:

El Centro de Operación de la Red está encargado de la gestión, administración y monitoreo de los enlaces y equipos.

El Centro de Asistencia Técnica tiene como función atender y solucionar problemas de carácter técnico en los equipos centrales de Red.

El Centro de Información de la Red tiene como objetivo proporcionar y mantener información técnica acerca de RedUNAM e Internet utilizando los servicios propios de la misma red.

El Departamento de Administración de Servidores mantiene funcionando en óptimas condiciones los servicios de correo, ftp, y www entre otros.

El Departamento de Atención a Usuarios, en coordinación con la Subdirección de Redes cuenta con un laboratorio para auxiliar a los usuarios finales de RedUNAM que utilizan una conexión vía módem.

Por otra parte la Coordinación de Conectividad tiene como objetivo mantener en óptimo funcionamiento todo lo relacionado con los elementos físicos de RedUNAM, se ocupa del mantenimiento de equipos de red y medios de transmisión principalmente.

· Conexión usando Trumpet Winsock:

Para hacer uso de los servicios de RedUNAM-Internet, es necesario primero configurar el programa Trumpet Winsock. Este programa se encuentra en la ventana RedUNAM/Internet, representado por un ícono en forma de trompeta subtitulado Marcar/Colgar. Dé un doble click en el ícono correspondiente para iniciarlo.

La primera vez que se ejecuta, el programa aparece minimizado en la parte inferior de la pantalla, para visualizar la ventana del mismo debe dar un doble click otra vez en este ícono.

Primeramente es necesario configurar algunos parámetros importantes relacionados con su módem. Para hacer esto despliegue el menú FILE y seleccione la opción SETUP.

En este momento aparece un cuadro de diálogo. Para configurar el puerto y la velocidad del módem dé un click en el botón DIALLER SETTINGS...

En este cuadro de diálogo aparecen dos listas desplegables para selección de puerto y velocidad. Haga un click en la lista desplegable 'COMM port' para seleccionar el puerto serie de comunicación correcto.

Los módems externos usualmente se conectan al los puertos COM1 ó COM2, en tanto que los módems internos se configuran como COM3 ó COM4. Para saber el número de puerto correcto debe consultar su manual o preguntar al proveedor que instaló su módem. En el último de los casos, puede repetir este proceso de configuración probando con todos los puertos COM, alguno de estos debe funcionar sin problemas. Seleccione dando un click en alguno de la lista, en este ejemplo elegiremos el puerto COM2.

Ahora hay que especificar la velocidad del módem, para esto dé un click en la lista desplegable 'Baud rate'.

Si no sabe la velocidad máxima seleccione entre 9600 y 19200 baudios. En este ejemplo seleccionaremos 19200 baudios.

A continuación dé un click en el botón OK.

Para que el programa guarde la configuración, ahora debe dar un click en el botón OK de este cuadro de dialogo.

El programa nos indica que es necesario reiniciarlo -es decir, salir y ejecutarlo de nuevo- para que utilice la configuración que definimos. Pero podemos definir algunos parámetros más antes de salir, así que aceptamos la indicación y continuamos...

Solo nos resta introducir su clave y los teléfonos de acceso, despliegue el menú DIALLER y seleccione la opción PROFILE.

En este momento hay que consultar la hoja de inscripción.

En la hoja se indican claramente la clave y el password de acceso a RedUNAM.

Es necesario escribir la clave de acceso a RedUNAM en el campo USERNAME de esta ventana.

A continuación seleccionamos el segundo campo moviendo el puntero con el mouse y dando un click.

En este campo hay que escribir el password de acceso a RedUNAM, ahora no verá lo que escribe, solo el número de caracteres, pero respete las mayúsculas y minúsculas.

Finalmente seleccionamos el campo de teléfonos: PHONE.

En su hoja de inscripción aparecen los teléfonos de acceso que se muestran a continuación.

Puede escribir hasta cuatro teléfonos separados por un espacio en blanco, en este caso escribiremos dos...

Primer teléfono...

espacio...

segundo teléfono.

Finalmente hay que dar un click en el botón OK.

Ahora sí, para que el programa acepte y utilice los cambios de configuración hay que salir y volver a entrar, despliegue el menú FILE y seleccione EXIT.

Para iniciar de nuevo el programa, otra vez debe dar un doble click en el ícono Marcar-Colgar.

Ahora que el programa está configurado, podemos establecer la conexión con RedUNAM. Despliegue el menú DIALLER y seleccione LOGIN.

Si existe algún problema físico con su módem, después de unos momentos aparecerá un mensaje de error; este indica que el puerto COM fué mal elegido, que el módem no está conectado correctamente, o que la conexión con la línea telefónica está fallando.

Si todo va bien el módem marcará al teléfono de RedUNAM, generalmente usted escuchará una serie de sonidos de marcado y conexión como estos...

En este momento se ha conectado. Recuerde que si el primer teléfono está ocupado no habrá conexión pero el programa intentará con los teléfonos alternativos.

Ahora el programa envía la clave de acceso.

Posteriormente envía el password.

La clave y el password son correctos, así que el proceso continúa.

En este momento se ha establecido la conexión a Internet, puede dar un click en el botón de minimización de la ventana para usar las aplicaciones.

· Telnet:

Para comprobar el buen funcionamiento de la conexión utilizaremos una aplicación de Internet simple llamada: Telnet. Para iniciar esta aplicación de Internet, debe dar un doble click en el ícono subtitulado: Telnet.

Telnet es una aplicación que nos permite tomar el control de alguna computadora conectada a Internet, también llamada 'host'. Ahora se nos pide el nombre de algún 'host', en este caso escribiremos el nombre de un servidor de correo electrónico de la UNAM: servidor.unam.mx

Para acceder a este 'host' dé un click en el botón OK.

Si después de unos segundos aparece una pantalla como esta, la conexión a Internet funciona correctamente y usted podrá utilizar cualquier otra aplicación sin ningún problema. En este caso el servidor nos pide una clave de acceso, si usted tramitó clave de correo electrónico es el momento de verificarla. Una vez más debe consultar su hoja de inscripción.

En la hoja se muestran la clave y el password de correo electrónico, es posible que estas claves sean iguales a las de acceso a RedUNAM, pero esto no es problema.

Ahora escriba la clave de correo electrónico. Si usted no tiene clave y password de correo, escriba cualquier cosa, más adelante se indica como salir de este programa.

Al terminar presione la tecla ENTER.

A continuación escriba el password de correo. En este caso usted no podrá ver lo que escribe en la pantalla, pero es necesario que escriba correctamente y respete las letras mayúsculas y minúsculas.

Nuevamente presione la tecla ENTER.

Si la clave y el password son correctos, aparecerá un mensaje relacionado con el sistema de este servidor y la línea de comandos con la leyenda: servidor%

En este momento hemos iniciado una sesión en el servidor. Obviamente, si usted no cuenta con una clave de correo, no podrá acceder, de cualquier modo, lo importante es que la conexión a Internet funciona correctamente. Puede salir del programa Telnet desplegando el menú FILE y seleccionando EXIT.

El programa pide confirmar la salida del mismo debido a que aún estamos en sesión, dé un click en el botón SÍ.

· E-Mail:

Antes de continuar verifique que el programa Trumpet se encuentre activo y minimizado.

Eudora es un programa especialmente diseñado para el manejo del correo electrónico. Para iniciarlo haga un doble click en este ícono.

La primera vez que lo ejecuta debe realizar una configuración mínima para que el programa funcione.

Primeramente debe escribir su dirección de correo electrónico en el cuadro de texto: POP account, para ello consulte nuevamente la hoja de inscripción y busque la clave y el password de correo.

Ahora escriba su clave de correo.

A continuación escriba: @servidor.unam.mx o bien el nombre del servidor de correo que usted use.

Esta es su dirección de correo electrónico, que será utilizada para el intercambio de correspondencia. Si desea que toda la correspondencia que usted envíe aparezca firmada con su nombre, dé un clic en el cuadro de texto REAL NAME y escriba su nombre completo.

Con esta configuración mínima el programa funcionará sin problemas. Más adelante se mencionarán otras opciones importantes.

Ahora usted puede utilizar el programa EUDORA para consultar su correo nuevo, para esto haga un click en el ícono: Check Mail.

Cada vez que usted intente consultar su correo nuevo, el programa pedirá su password de correo electrónico, escríbalo respetando mayúsculas y minúsculas. Recuerde que no podrá ver lo que escribe.

Dé un click en el botón OK.

En este momento el programa intentará conectarse al servidor para transportar el correo hasta su computadora.

Ahora el programa notifica que usted tiene nuevo correo, haga un clic en el botón OK para continuar.

Al fondo aparece una ventana llamada IN que muestra todos los correos que han llegado, usted debe tener al menos un correo nuevo. Note que los correos nuevos tienen una marca del lado izquierdo. Para ver este nuevo correo haga un doble clic en la línea correspondiente.

En este momento podría leerlo, pero por ahora vamos a continuar, no se preocupe, el correo permanecerá allí aunque usted salga del programa. Cierre la ventana de este correo haciendo un clic en el botón de control y seleccione la opción CERRAR.

Ahora vamos a enviar un correo de prueba, para esto dé un clic en el ícono: New Message.

A continuación escriba la dirección destino.

Seleccione el campo SUBJECT haciendo un clic con el mouse y escriba una breve descripción del contenido del mensaje.

Nuevamente use el mouse para finalmente escribir el contenido de la carta. Sea breve, es solo un correo de prueba.

En algunas ocasiones usted necesitará enviar además del texto de su carta, un archivo ejecutable, un gráfico, un documento con formato, etc. Es decir; archivos binarios. Para esto utilizaremos la opción adjuntar, haga un clic en el ícono: ATTACHMENT.

A continuación aparece un selector de archivos, como esta es solo una prueba seleccione un archivo pequeño, no mayor de 10,000 bytes.

Clic en el botón: Attach para continuar.

El usuario destino recibirá ambos, el texto del correo y el archivo binario. Usted puede enviar más de un archivo adjunto haciendo nuevamente clic's en el ícono: Attachment y seleccionando otros, pero por ahora solo enviaremos uno. Para enviar este correo haga un clic en el botón SEND.

En este momento el programa intentará enviar el correo...

Cada vez que usted se conecta, ocupa una parte del tiempo de conexión en la red, pero cuando algún usuario no utiliza este recurso por más de 10 minutos, automáticamente es desconectado. Al escribir un correo con Eudora usted no utiliza el recurso que otro usuario podría aprovechar, así que de presentarse algún error al tratar de enviar el correo, lo más probable es que usted haya sido desconectado. Pero no hay de que preocuparse, continúe con el proceso. Ahora haga un clic en el ícono: Mail-OUT.

De acuerdo a la configuración avanzada por omisión, la ventana OUT debe contener todos los correos que usted escribió hayan sido enviados o no. Esto significa que usted puede escribir su correo antes de conectarse a RedUNAM/Internet, y una vez conectado, enviarlo haciendo un clic en la línea correspondiente de la ventana OUT.

A continuación realizaremos una configuración más avanzada. Para modificar las opciones de configuración seleccione del menú: TOOLS la opción: OPTIONS

Otra vez aparece el cuadro de diálogo de configuración inicial, aquí puede cambiar la configuración mínima si lo cree conveniente. Pero ahora modificaremos otras opciones importantes haciendo un clic en el ícono: Checking Mail.

Cuando usted necesite verificar su correo nuevamente, el programa pedirá su password para entrar al servidor, al activar la opción: Save password, no será necesario escribirlo cada vez que se conecte a RedUNAM/Internet. Pero esto puede no ser conveniente por razones de seguridad, pues cualquier persona que utilice su computadora y active Eudora, podrá ver su correo nuevo.

Si realmente necesita utilizar otras computadoras además de la suya para leer su correo, active la opción: Leave mail on server, esto permitirá que una copia de su correo permanezca en el servidor, pero ello implica que usted tendrá que hacer una labor de limpieza periódica eliminando correos antiguos, de lo contrario el espacio en el servidor se saturará y tendrá problemas.

Haga un clic en el botón OK para aceptar estos cambios de configuración.

Para salir de Eudora seleccione la opción EXIT del menú FILE. Todos los correos serán almacenados y usted los podrá leer o enviar la próxima vez que entre al programa.

FTP:

Antes de continuar verifique que el programa Trumpet se encuentre activo y minimizado. Para ejecutar el programa, dé un doble clic sobre el ícono FTP.

La lista desplegable: PROFILE NAME indica el nombre de algún acceso predefinido, en este caso la configuración por omisión permitirá traer a su máquina el programa NETSCAPE desde un servidor de la UNAM. Una vez que sepa manejar el programa podrá hacer un clic en esta lista desplegable para elegir otras configuraciones y acceder a otros servidores con información de interés.

En caso de que usted desee acceder algún servidor que no se encuentre en esta lista, puede configurar el acceso posicionándose en los otros campos y haciendo clic's con el mouse para escribir o seleccionar.

En el campo o cuadro de texto: HOST NAME puede escribir el nombre o dirección IP del servidor al que usted desee acceder.

La lista desplegable: HOST TYPE debe contener la opción: -auto detect-

Los campos USER ID y PASSWORD se utilizan para indicar la clave y contraseña de acceso a este servidor. En caso de no conocerlas puede intentar acceder de forma anónima, esto es utilizar la palabra: "anonymous" como clave y su dirección de correo como contraseña.

En este caso accederemos de forma anónima a un servidor de la UNAM.

El campo: Remote Host contiene el directorio inicial de entrada al servidor, esto permitirá un cambio inmediato de directorio remoto

al establecer la conexión.

El campo Local PC indica el directorio de su PC que se usará para enviar y recibir por omisión.

No es necesario escribir algo en estos campos, usted podrá elegir los directorios cuando establezca la conexión.

Para establecer esta conexión haga un clic en el botón: OK

Una vez que se ha establecido la conexión aparecen los directorios y archivos correspondientes en la ventana principal. A la izquierda hay una lista del del directorio local, es decir el directorio de su PC, en la parte derecha se presentan los archivos correspondientes al sistema remoto, es decir al servidor de la UNAM en este caso.

Naturalmente usted puede cambiar de directorio remoto ó local cuando lo desee, los directorios aparecen en la parte de arriba y usted puede dar un doble clic en cualquiera de estos para accederlos. El directorio definido como .. permite acceder al directorio anterior. Ahora vamos a buscar el programa Netscape en su versión para PC, así que dé un clic en el directorio remoto: pc.

Una vez en el directorio de programas para PC, sabemos que necesitamos un programa relacionado con RedUNAM, así que *entraremos al directorio RedUNAM haciendo un doble clic otra vez.*

Sabemos que Netscape es una aplicación que funciona bajo windows, así que es necesario entrar al directorio windows, una vez más, doble clic para entrar.

En el directorio windows debemos buscar un subdirectorio relacionado con el Web, Navegación o WWW. El directorio es muy grande, así que para buscar hay que hacer algunos clic's en las flechas de deslizamiento.

Existen muchas aplicaciones, pero la que nos interesa se encuentra en el subdirectorio www. Haremos un doble clic final para entrar a éste.

Es aquí donde se encuentra el subdirectorio netscape, entraremos con doble clic para elegir la versión más adecuada.

Existen muchas versiones de este programa, las últimas son mejores pero requieren más recursos. Los nombres generalmente tienen un número 32 ó 16 que indica el número de bits del sistema operativo que debe usarse. Windows 95 y posteriores pueden usar los programas de 32 bits, las versiones anteriores de Windows generalmente sólo manejan 16 bits.

Teóricamente las versiones de 32 bits deberían ser más rápidas, pero en una conexión normal vía módem no se aprecia diferencia alguna en velocidad. Recomendamos elegir una versión de 16 bits puesto que funcionará en cualquier windows.

Antes de continuar es necesario elegir un directorio dentro de su PC en donde será almacenado el programa Netscape, en este caso crearemos un subdirectorio dentro del directorio del programa FTP. Para esto haga un clic en el botón: MkDir

Para llevar un órden podemos llamar a este subdirectorio: netscape, escríbalo en este cuadro de texto.

Clic en el botón: OK para continuar.

Una vez creado hay que entrar al subdirectorio antes de efectuar la transferencia, haga un doble clic sobre éste.

Para elegir el archivo a traer es de utilidad verificar el tamaño del mismo, haga un clic en el botón: DirINFO.

A la izquierda de los nombres se encuentra el tamaño en bytes y otros datos, considere que un archivo de 1 Megabyte tardará aproximadamente 10 minutos en llegar usando un módem de 28800 bps.

Cierre la ventana del block de notas y seleccione el archivo que usted quiera traer haciendo un clic sobre el nombre del mismo.

Para comenzar la transferencia haga un clic en la flecha hacia la izquierda. Si en un momento dado el proceso tarda mucho, usted puede detener la transferencia haciendo un clic en el botón: Cancel, no se preocupe, el programa Netscape no es realmente esencial.

Una vez que el archivo ha llegado completo, o bien si el proceso fué cancelado, podemos cerrar la conexión al servidor, haga un clic en el botón: Close.

No se preocupe si el archivo está incompleto porque detuvo la transmisión o surgió otro problema, simplemente continúe observando el proceso de instalación de Netscape por si alguna vez desea intentarlo de nuevo.

Los archivos almacenados en un servidor generalmente están compactados y empaquetan varios archivos en uno solo para ahorrar tiempo de transmisión. Si el archivo ha llegado completo podemos descompactarlo, en este caso simplemente hay que ejecutarlo y esto lo podemos hacer desde la ventana del programa FTP. Haga un clic en el nombre del archivo.

Y otro clic en el botón: Exec.

En estos momentos el programa se descompactará en diversos archivos, pero el archivo transferido originalmente permanecerá intacto.

Para observar los nuevos archivos generados haga un clic en el botón: Refresh.

Ahora hay que buscar algún archivo que permita instalar Netscape, por ejemplo: install.exe o bien: setup.exe, para esto use las flechas y barras de deslizamiento.

Finalmente encontramos el archivo: setup.exe, que servirá para instalar el programa Netscape, lo único que debe hacer es ejecutarlo, y esto es posible desde el administrador de archivos usando la opción: Ejecutar del menú: Archivo. O bien desde el mismo FTP, simplemente seleccionando el archivo con un clic...

...y haciendo otro clic en el botón: Exec.

Le recomendamos que por ahora cancele la operación y continúe con el programa FTP, pues la instalación lleva tiempo y posiblemente usted sea desconectado. Podrá instalar Netscape una vez que esté fuera de línea, es decir, desconectado de RedUNAM, para que no desperdicie su tiempo de conexión.

No se preocupe si no pudo llegar hasta este punto, de cualquier forma RedUNAM distribuye una versión del navegador: Internet Explorer, con el cual usted puede consultar la mayor parte de las páginas en Internet.

Para salir del programa haga un clic en el botón: Exit.

· WWW:

Antes de continuar, verifique que el programa Trumpet se encuentre activo y minimizado.

Para ejecutar el navegador: Internet Explorer haga un doble clic en este ícono.

Automáticamente el navegador intentará acceder a la página configurada por omisión, en este caso es una página de Microsoft.

Si la lectura de una página es muy lenta, o la navegación se bloquea debido a problemas de acceso, usted puede hacer un clic en el ícono: STOP para intentar acceder a otra dirección. En este caso recomendamos que espere al menos 30 segundos.

Finalmente, si la página fue leída en su totalidad o se usó el ícono STOP, el navegador regresa a su estado normal de espera. Recuerde que cada página tiene una dirección que aparece en el cuadro de texto mostrado enseguida.

Usted puede hacer un clic sobre este cuadro de texto y escribir cualquier dirección para acceder a la página correspondiente, pero hay formas de acceso más simples.

Una vez que sepa viajar con soltura en la red, encontrará algunas páginas de profundo interés cuya información es actualizada constantemente. No es necesario escribir la dirección completa cada vez que desee acceder a estas, usted puede guardar las direcciones en una lista llamada: Hot-List ó favoritos, en este caso. Y de esta forma la próxima vez que se conecte, simplemente bastará con elegir la que desee haciendo un clic.

Para agregar una dirección a esta lista debe hacer un clic en el ícono aquí mostrado, en el momento en que la página elegida se encuentre en pantalla.

En esta lista ya existen algunas páginas así que le demostraremos como acceder a estas, dé un clic en el ícono: Hot-List.

Esta ventana contendrá todas las direcciones que usted agregue a la lista, por ahora solo existe la página de la UNAM, accederemos a ésta haciendo un clic en el botón: Saltar a.

La página de la UNAM como muchas otras no cabe completa en pantalla, así que usted deberá usar las barras de desplazamiento haciendo algunos clic's para ver el resto de la página.

En cada página existen ligas que nos permitirán navegar hacia otras páginas, de esta forma podemos pasar de un tema a otro. Estas ligas están marcadas con un color distinto, subrayado o bien son mostradas en forma de gráficos, en este caso, la página de la UNAM tiene ambas, por ejemplo, para acceder a la página: Acervos de Información existen dos ligas, una en forma gráfica y otra como texto subrayado.

Haremos un clic en esta liga.

En este momento aparece otra página que puede contener gráficos y textos, es así como usted navegará entre las páginas de Internet y de esta forma obtendrá toda la información que necesite.

De acuerdo a la configuración por omisión del programa, todas las páginas que usted va consultando son almacenadas en su disco duro, por lo tanto usted puede accederlas casi instantáneamente sin necesidad de leerlas nuevamente de la red. A esto se le denomina memoria caché.

De esta forma, cada vez que usted se conecte a Internet e inicie una nueva sesión con el programa navegador, esta se agilizará si usted accede a páginas que en su mayoría se han leído en sesiones previas.

Por supuesto, algunas páginas son actualizadas constantemente, y por lo tanto es necesario también actualizarlas en la memoria caché,

por este motivo existe el ícono: Reload.

Al hacer un clic sobre éste, el navegador intentará releer la página directamente del servidor. Incluso si alguna vez tiene problemas de acceso con alguna página, también puede usar este ícono y es muy posible que en intentos posteriores tenga éxito.

En cada sesión, el programa navegador guarda una secuencia de acceso, esto permite acceder de forma inmediata a cualquiera de las páginas leídas durante la misma.

Si en algún momento usted desea regresar a la página anterior puede hacer un click en el ícono con la flecha hacia la izquierda. De esta forma usted puede regresar de ser necesario hasta la primera página pasando por todas las intermedias. Por ahora solo haga un clic en este ícono.

Desde luego, si usted puede ir a una página anterior, también puede regresar a la siguiente. Haga un clic en el ícono con la flecha hacia la derecha para observar este efecto.

Nuevamente usted se encuentra en la última página que fue leída de la red.

En un momento dado usted deseará leer con más detenimiento, conservar o imprimir alguna página individual, lo más conveniente es hacerlo fuera de línea para no malgastar tiempo de conexión. Para ello podemos grabar la página en el disco duro con un nombre determinado: despliegue el menú ARCHIVO y seleccione la opción GUARDAR COMO.

Es importante entender que lo único que se grabará de esta forma es la estructura y el texto de la página, no así los gráficos.

A continuación aparece un selector de archivos y usted podrá escribir el nombre con el que desea salvar esta página. Teclee el nombre, en este caso: acervos.htm y haga un clic en el botón ACEPTAR.

Si usted desea grabar algún gráfico, posicione el puntero sobre éste y presione el botón intermedio o derecho del mouse.

De esta forma aparece un menú extra que le permite guardar el gráfico seleccionado en disco y además hacer otras operaciones que usted sabe manejar.

Una vez que termine su sesión con el navegador puede salir desplegando el menú ARCHIVO y seleccionando la opción SALIR.

· Desconexión usando Trumpet:

Ahora usted puede desconectarse de RedUNAM/Internet, para esto hay que indicarle al módem que libere la línea telefónica. Haga un doble click en el ícono de Trumpet para maximizarlo, despliegue el menú: DIALLER y seleccione la opción: BYE.

Después de unos momentos aparecerá el mensaje de desconexión exitosa y usted podrá salir del programa Trumpet desplegando el menú FILE y seleccionando la opción EXIT.

De esta forma, para conectarse de nuevo a RedUNAM necesitará primero ejecutar el programa Trumpet, minimizarlo y posteriormente abrir las aplicaciones de Internet.

Cuando desee terminar simplemente cierre las aplicaciones y libere la línea telefónica desde el programa Trumpet.

· Conexión/Desconexión usando Windows 95/98:

El sistema Windows 95, correctamente instalado, contiene su propio socket para la conexión a una red de comunicaciones funcionando bajo el protocolo TCP/IP. Esto significa que usted puede manejar las aplicaciones de Internet sin necesidad de usar el programa: Trumpet Winsock.

Primero hay que instalar el protocolo TCP/IP, es muy posible que su computadora ya cuente con este protocolo preinstalado, sin embargo, continúe observando el proceso por si en un momento dado lo necesita.

Para instalar el protocolo haga un doble clic en el ícono: Mi PC

En esta ventana busque el icono: Panel de Control y haga un doble clic sobre el mismo.

La ventana del Panel de Control nos permite configurar diversos aspectos del equipo, en este caso haga un doble clic sobre ícono: RED

El siguiente cuadro de diálogo muestra algunas opciones de configuración, pero primero necesitamos agregar el protocolo TCP/IP, así que haga un clic en el botón: Agregar.

Ahora seleccione la opción: Protocolo y haga un clic en el botón: Agregar.

El protocolo que agregaremos es fabricado por microsoft, seleccione el nombre de esta compañía que aparece del lado izquierdo haciendo un clic.

Aquí aparecen los posibles protocolos de esta compañía, así que seleccione el que ya hemos mencionado: TCP/IP y haga un clic en el ícono: Aceptar.

Observe la lista de elementos de red, el protocolo: TCP/IP debe aparecer como un elemento más.

Ahora podemos cerrar la ventana del panel de control pues ya no la necesitaremos.

La ventana: Mi PC, debe contener un folder llamado: Acceso telefónico a redes, haga un doble clic sobre éste.

En la ventana de acceso telefónico encontrará una aplicación para realizar conexiones, la ejecutaremos haciendo un doble clic sobre este ícono.

La nueva conexión debe tener un nombre, pero debido a que cada conexión únicamente maneja un teléfono, es muy probable que en el futuro usted desee generar más conexiones con distintos teléfonos, así que a esta la llamaremos: RedUNAM1, Escriba en el cuadro de texto y haga un clic en el botón: Siguiente >.

En esta página es necesario indicar el código de país, haga un clic en la lista desplegable y busque México con ayuda de la barra de desplazamiento. Cuando lo encuentre haga un clic para seleccionarlo.

A continuación escribiremos el número telefónico, para esto haga un clic en el cuadro de texto correspondiente y escriba algún teléfono de RedUNAM. Cuando termine haga un clic en el botón: Siguiente >.

Finalmente se nos indica que la conexión se ha generado y solamente resta hacer un clic en el botón: Finalizar.

Ahora observe que la ventana de acceso telefónico contiene el ícono: RedUNAM1 que indica la conexión que acabamos de generar, pero aún debemos configurar algunos detalles así que haga un clic en este ícono y vaya al menú: Archivo. En este menú seleccione la opción: Propiedades.

Esta ventana de configuración contiene algunos datos que previamente se introdujeron pero hay que indicar otros, haga un clic en la pestaña: Tipo de servidor.

En la parte de arriba hay una lista desplegable con la opción: PPP por defecto, de cualquier modo verifique que ésta se encuentre activa.

A continuación puede desactivar el protocolo de Red NetBEUI para evitar algunos inconvenientes durante el proceso de conexión.

Ahora hay que configurar el protocolo TCP/IP para esto dé un clic en el botón: Configuración TCP/IP.

En esta nueva ventana verifique la opción que indica: Dirección IP asignada por el servidor, debe estar activa.

Posteriormente seleccione la opción: Direcciones de servidor asignadas por el usuario. El servidor DNS permite hacer la conversión de nombres a direcciones IP, haga un clic sobre este cuadro de texto y escriba la dirección: 132.248.237.250

A continuación escriba la dirección secundaria: 132.248.10.2 y dé un clic en el botón: Aceptar.

Finalmente, para guardar los datos que han sido introducidos haga un clic en el botón: Aceptar de la ventana principal.

Ahora la conexión está totalmente configurada y usted puede utilizarla, para esto asegúrese de que el módem este encendido y conectado correctamente y haga un doble clic sobre el ícono de conexión, en este caso RedUNAM1.

En esta ventana escriba su clave y password de acceso a TROUTER.

Para establecer la conexión finalmente haga un clic en el botón: Conectar.

6.2 Selección de los formatos de animación y sonido que se utilizarán en la estructura final

En cuanto a los formatos de audio, no hay mucho de que preocuparse, pues existen muchos programas gratuitos que soportan todos los formatos de música y audio existentes: MIDI, MOD, MPG-3, WAV, AIFF, etc. y pueden enlazarse con el programa principal sin mayores problemas. Además es posible hacer conversiones entre algunos de estos.

Por otra parte, los formatos más populares de video y animación son también soportados, pero aquí es necesario considerar las características generales de velocidad/compresión/calidad.

mpeg y derivados (avi 8/16/24, mov 8/16/24) formatos de compresión delta (GIF, FLI, ANIM 8/16/32) formatos de compresión delta NTSC (ANIM-HAM 8/16/32)

Se obtuvieron los siguientes resultados para los formatos más adecuados:

Formato	Compresión	Velocidad	Núm. de colores
MPEG	Alta	Baja	TrueColor
AVI/MOV16	Media	Media	65535
ANIM-HAM	Media	Alta	TrueColor
ANIM	Baja	Alta	256
GIF	*Baja*	*Media*	*256*

Desde luego, los formatos que ofrecen 65535 colores o más, tienen que degradar las imágenes ligeramente dependiendo del radio de compresión, mientras que los formatos a 256 colores no pierden detalle durante la compresión pero no ofrecen buenos resultados para imágenes de calidad fotográfica.

De cualquier modo, salta a la vista el formato ANIM-HAM (Hold and Modify) exclusivo de esta plataforma, por ser el que permite el despliegue más rápido (60 Hz en una pantalla NTSC completa) en color real. Este formato y los otros similares ANIM 8/16/32 fueron utilizados para componer las animaciones parciales de este trabajo.

6.3 Elaboración de las animaciones y recursos de programación necesarios para proveer de elementos explicativos y didácticos al usuario

Para esta parte fue necesaria la captura de imágenes correspondientes a los programas utilizados en Windows para Internet y la realización de imágenes auxiliares, didácticamente hablando, para el resto del sistema. Estas imágenes se realizaron y adecuaron utilizando programas para diseño en 2 y 3 dimensiones. Después se procesaron para obtener animaciones parciales en los formatos elegidos.

6.4 Realización de un programa para enlazar las imágenes y animaciones parciales, así como los segmentos de audio y música en una estructura multimedia final

Para realizar una estructura multimedia existen muchos programas comerciales que están muy bien programados y son muy rápidos, sin embargo, ninguno soporta directamente todos los formatos de audio y animación que utilizaremos y la mayoría no permiten enlaces óptimos mediante el sistema operativo para establecer una comunicación con otros programas. Además no aprovechan al 100% el entorno multitarea y tienen un costo relativamente alto.

Existen también sistemas orientados a objetos que simplifican mucho el trabajo, desafortunadamente se venden a precios muy altos y los resultados obtenidos, inclusive con los procesadores más rápidos (equivalentes a un Pentium) de hoy en día, no son realmente aceptables para obtener un resultado de calidad en forma de video continuo.

En este sentido, solamente podemos utilizar los programa gratuitos del dominio público o un programa comercial con un costo realmente económico y programar el resto.

6.4.1 Aspectos que debe cubrir el programa o planteamiento del problema

Afortunadamente, hace algunos meses se distribuyó, con una revista especializada -AmigaCU, Feb 98- en esta plataforma, una versión en CD de uno de los programas multimedia más avanzados para esta arquitectura, llamado ScalaMM.

Esta versión gratuita del programa ScalaMM tiene las siguientes características importantes a considerar para nuestro trabajo:

Ventajas:

-Trabaja perfectamente en modo multitarea.

-Es posible utilizar una gama muy amplia de eventos gráficos, así como efectos digitales de video en tiempo real de alta calidad, subtítulos, etc.

-Cuenta con un lenguaje especial muy simple que permite programar el flujo de los eventos gráficos de acuerdo a condiciones de entrada: menús, teclas, botones, etc.

Desventajas:

-Desafortunadamente los comandos para la intercomunicación estándar con otras aplicaciones no están documentados y esto es una gran limitante pues el número de formatos de audio y video soportados por esta versión del programa es muy reducido.

-El flujo de los eventos gráficos es totalmente lineal, es decir, el programa no lee la siguiente imagen/animación hasta que termina el despliegue o efecto de video actual, esto puede generar pausas entre animaciones que pueden ser inaceptables para una didáctica adecuada, inclusive si se utiliza la opción de precargado en memoria o caché.

-No explota a fondo las capacidades gráficas de la arquitectura.

En este orden de ideas y sabiendo que no es necesario reinventar la rueda, podemos deducir que el camino más adecuado es realizar un nuevo PROGRAMA COMPLEMENTO que resuelva las limitantes o problemas del programa ScalaMM.

De esta forma, podemos utilizar un script del programa ScalaMM como base de la estructura multimedia, pero este script deberá servir para hacer llamadas a nuestro programa y así solventar las limitantes mencionadas.

6.4.2 Solucionando el problema

6.4.2.1 Comunicación con el programa ScalaMM

Nuestro programa y el programa ScalaMM deben correr simultáneamente, de este modo podemos utilizar las capacidades de ambos programas en el momento que se indique. Para lograr esto debemos establecer una comunicación entre el script del programa ScalaMM y nuestro programa.

Si no es posible utilizar el sistema de comunicación estándar desde el programa ScalaMM entonces debemos buscar otra forma de hacerlo, para esto consideramos lo siguiente:

El sistema operativo se encarga de administrar todos los recursos del sistema, por lo tanto, siempre debe estar enterado de las instrucciones enviadas al hardware de audio o de video, sabemos que la arquitectura AMIGA tiene una gran flexibilidad en cuanto a los despliegues gráficos, pues puede crear pantallas de cualquier tamaño, todo esto programando al coprocesador COPPER para liberar al CPU. Afortunadamente el programa ScalaMM permite esta flexibilidad para crear diversos tamaños de pantalla, así que de momento se nos ocurre pensar en que podemos detectar el tamaño de la pantalla actual, activa o prioritaria (pues puede haber muchas pantallas virtuales) leyendo el programa del coprocesador COPPER.

En este sentido, podemos hacer que el programa ScalaMM despliegue una pantalla de tamaño poco probable para cualquier otra aplicación, 16x1 por ejemplo, que además ahorra memoria gráfica, y utilizar los registros de color para almacenar una señal que podrá ser interpretada por nuestro programa. Así el problema se reduce a detectar el momento en que está "pantalla señal" es desplegada o activada.

Para realizar lo anterior es necesario saber en donde se encuentra el código COPPER de la pantalla activa o actual, así que necesitamos echar un vistazo a las estructuras del sistema operativo

Estructuras de despliegue gráfico del sistema operativo.

La estructura principal o base se define a continuación en el archivo a incluir para programar la arquitectura en lenguaje C: graphics/gfxbase.h

```
#ifndef GRAPHICS_GFXBASE_H
#define GRAPHICS_GFXBASE_H
/*
**      $VER: gfxbase.h 39.21 (21.4.93)
**      Includes Release 40.15
**
**      graphics base definitions
**
**      (C) Copyright 1985-1993 Commodore-Amiga, Inc.
**          All Rights Reserved
*/

#ifndef EXEC_LISTS_H
#include <exec/lists.h>
#endif
#ifndef EXEC_LIBRARIES_H
#include <exec/libraries.h>
#endif
#ifndef EXEC_INTERRUPTS_H
#include <exec/interrupts.h>
#endif
#ifndef GRAPHICS_MONITOR_H
#include <graphics/monitor.h>
#endif

struct GfxBase
{
        struct  Library  LibNode;
        struct  View *ActiView;
        struct  copinit *copinit;       /* ptr to copper start up list, */
        LONG    *cia;                   /* for 8520 resource use */
        LONG    *blitter;               /* for future blitter resource use */
        UWORD   *LOFlist;
        UWORD   *SHFlist;
        struct  bltnode *blthd,*blttl;
        struct  bltnode *bsblthd,*bsblttl;
        struct  Interrupt vbsrv,timsrv,bltsrv;
        struct  List     TextFonts;
        struct  TextFont *DefaultFont;
        UWORD   Modes;                  /* copy of current first bplcon0 */
```

```
BYTE    VBlank;
BYTE    Debug;
WORD    BeamSync;
WORD    system_bplcon0;         /* it is ored into each bplcon0 for display */
UBYTE   SpriteReserved;
UBYTE   bytereserved;
UWORD   Flags;
WORD    BlitLock;
WORD    BlitNest;

struct  List BlitWaitQ;
struct  Task *BlitOwner;
struct  List TOF_WaitQ;
UWORD   DisplayFlags;           /* NTSC PAL GENLOC etc*/
                                /* flags initialized at power on */
struct  SimpleSprite **SimpleSprites;
UWORD   MaxDisplayRow;          /* hardware stuff, do not use */
UWORD   MaxDisplayColumn;       /* hardware stuff, do not use */
UWORD   NormalDisplayRows;
UWORD   NormalDisplayColumns;
/* the following are for standard non interlace, 1/2 wb width */
UWORD   NormalDPMX;             /* Dots per meter on display */
UWORD   NormalDPMY;             /* Dots per meter on display */
struct  SignalSemaphore *LastChanceMemory;
UWORD   *LCMptr;
UWORD   MicrosPerLine;          /* 256 time usec/line */
UWORD   MinDisplayColumn;
UBYTE   ChipRevBits0;
UBYTE   MemType;
UBYTE   crb_reserved[4];
UWORD   monitor_id;
ULONG   hedley[8];
ULONG   hedley_sprites[8];      /* sprite ptrs for intuition mouse */
ULONG   hedley_sprites1[8];     /* sprite ptrs for intuition mouse */
WORD    hedley_count;
UWORD   hedley_flags;
WORD    hedley_tmp;
LONG    *hash_table;
UWORD   current_tot_rows;
UWORD   current_tot_cclks;
UBYTE   hedley_hint;
UBYTE   hedley_hint2;
ULONG   nreserved[4];
LONG    *a2024_sync_raster;
UWORD   control_delta_pal;
UWORD   control_delta_ntsc;
struct  MonitorSpec *current_monitor;
struct  List MonitorList;
struct  MonitorSpec *default_monitor;
struct  SignalSemaphore *MonitorListSemaphore;
VOID    *DisplayInfoDataBase;
UWORD   TopLine;
struct  SignalSemaphore *ActiViewCprSemaphore;
ULONG   *UtilBase;              /* for hook and tag utilities.
                                   had to change because of name clash */
ULONG   *ExecBase;              /* to link with rom.lib       */
UBYTE   *bwshifts;
UWORD   *StrtFetchMasks;
UWORD   *StopFetchMasks;
UWORD   *Overrun;
WORD    *RealStops;
UWORD   SpriteWidth;    /* current width (in words) of sprites */
UWORD   SpriteFMode;            /* current sprite fmode bits  */
BYTE    SoftSprites;  /* bit mask of size change knowledgeable sprites */
BYTE    arraywidth;
UWORD   DefaultSpriteWidth;     /* what width intuition wants */
BYTE    SprMoveDisable;
UBYTE   WantChips;
```

```
        UBYTE   BoardMemType;
        UBYTE   Bugs;
        ULONG   *gb_LayersBase;
        ULONG   ColorMask;
        APTR    IVector;
        APTR    IData;
        ULONG   SpecialCounter;         /* special for double buffering */
        APTR    DBList;
        UWORD   MonitorFlags;
        UBYTE   ScanDoubledSprites;
        UBYTE   BP3Bits;
        struct  AnalogSignalInterval MonitorVBlank;
        struct  MonitorSpec *natural_monitor;
        APTR    ProgData;
        UBYTE   ExtSprites;
        UBYTE   pad3;
        UWORD   GfxFlags;
        ULONG   VBCounter;
        struct  SignalSemaphore *HashTableSemaphore;
        ULONG   *HWEmul[9];
};

#define ChunkyToPlanarPtr HWEmul[0]

/* Values for GfxBase->DisplayFlags */
#define NTSC            1
#define GENLOC          2
#define PAL             4
#define TODA_SAFE       8
#define REALLY_PAL      16      /* what is actual crystal frequency
                                   (as opposed to what bootmenu set the agnus to)?
                                   (V39) */
#define LPEN_SWAP_FRAMES        32
                                /* LightPen software could set this bit if the
                                 * "lpen-with-interlace" fix put in for V39
                                 * does not work. This is true of a number of
                                 * Agnus chips.
                                 * (V40).
                                 */

#define BLITMSG_FAULT   4

/* bits defs for ChipRevBits */
#define GFXB_BIG_BLITS 0
#define GFXB_HR_AGNUS   0
#define GFXB_HR_DENISE 1
#define GFXB_AA_ALICE   2
#define GFXB_AA_LISA    3
#define GFXB_AA_MLISA   4       /* internal use only. */

#define GFXF_BIG_BLITS 1
#define GFXF_HR_AGNUS   1
#define GFXF_HR_DENISE 2
#define GFXF_AA_ALICE   4
#define GFXF_AA_LISA    8
#define GFXF_AA_MLISA   16      /* internal use only */

/* Pass ONE of these to SetChipRev() */
#define SETCHIPREV_A    GFXF_HR_AGNUS
#define SETCHIPREV_ECS  (GFXF_HR_AGNUS | GFXF_HR_DENISE)
#define SETCHIPREV_AA   (GFXF_AA_ALICE | GFXF_AA_LISA | SETCHIPREV_ECS)
#define SETCHIPREV_BEST 0xffffffff

/* memory type */
#define BUS_16          0
#define NML_CAS         0
#define BUS_32          1
#define DBL_CAS         2
```

```
#define BANDWIDTH_1X   (BUS_16 | NML_CAS)
#define BANDWIDTH_2XNML        BUS_32
#define BANDWIDTH_2XDBL        DBL_CAS
#define BANDWIDTH_4X   (BUS_32 | DBL_CAS)

/* GfxFlags (private) */
#define NEW_DATABASE   1

#define GRAPHICSNAME   "graphics.library"

#endif  /* GRAPHICS_GFXBASE_H */
```

Como se observa, la estructura 'GfxBase' tiene acceso a todas las estructuras gráficas importantes del sistema, por esto se utiliza al abrir y utilizar la librería graphics del sistema operativo.

Lo que nos interesa por ahora es encontrar la dirección de memoria en donde se almacena el código, en el formato del coprocesador COPPER, de la pantalla activa o actual, si observamos las primeras líneas de la estructura misma encontramos un puntero a la estructura 'View' llamado *ActiView, según los manuales del sistema, las estructuras de despliegue en pantalla son conocidas como vistas o *views*, así que es muy probable que encontremos aquí lo que buscamos.

```
...
struct GfxBase
{
      struct      Library  LibNode;
      struct      View *ActiView;
      struct      copinit *copinit; /* ptr to copper start up list*/
...
```

Antes de continuar, llama la atención la siguiente línea, en donde se define un puntero a la estructura 'copinit' que en realidad guarda estrecha relación con lo que buscamos, pero después de leer en el archivo de la estructura correspondiente, resulta que ésta es una estructura privada, exclusiva para el uso del sistema operativo y que en el futuro puede cambiar, por lo tanto, no debe tocarse. Así lo haremos.

A continuación observamos en el archivo: graphics/view.h la estructura 'View' misma para buscar algo relacionado con las instrucciones COPPER de despliegue:

```
...
struct View
{
        struct  ViewPort *ViewPort;
        struct  cprlist *LOFCprList; /* for interlaced and noninterlaced */
        struct  cprlist *SHFCprList;    /* only used during interlace */
        WORD    DyOffset,DxOffset;    /* for complete View positioning */
                                /* offsets are +- adjustments to standard #s
*/
        UWORD   Modes;                  /* such as INTERLACE, GENLOC */
};
...
```

Aquí podemos ver que efectivamente, existen dos punteros a la estructura 'cprlist' llamados *LOFCprList y *SHFCprList. El primero se utiliza para todo tipo de despliegues, el segundo solamente para despliegues entrelazados.

La estructura 'cprlist' parece ser una abreviatura de *Copper List*, esto no es mas que la lista de

instrucciones para el COPPER que permiten generar un despliegue gráfico. Ahora debemos buscar en el archivo: graphics/copper.h, que contiene todas las definiciones relacionadas con este coprocesador, la estructura 'cprlist':

```
...
/* structure of cprlist that points to list that hardware actually executes
*/
struct cprlist
{
    struct cprlist *Next;
    UWORD   *start;          /* start of copper list */
    WORD    MaxCount;        /* number of long instructions */
};
...
```

Finalmente, encontramos el puntero a la dirección de memoria en donde se encuentra la lista COPPER del despliegue actual o activo, de manera lógica, se llama *start y es un apuntador a una palabra sin signo (UnsignedWORD).

Por el camino, podemos observar los comentarios de la estructura privada 'copinit' anteriormente mencionada:

```
...
/* Private graphics data structure. This structure has changed in the past,
 * and will continue to change in the future. Do Not Touch!
 */
struct copinit
{
    UWORD vsync_hblank[2];
    UWORD diagstrt[12];      /* copper list for first bitplane */
    UWORD fm0[2];
    UWORD diwstart[10];
    UWORD bplcon2[2];
    UWORD sprfix[2*8];
    UWORD sprstrtup[(2*8*2)];
    UWORD wait14[2];
    UWORD norm_hblank[2];
    UWORD jump[2];
    UWORD wait_forever[6];
    UWORD   sprstop[8];
};
...
```

En este sentido, es posible deducir que a partir de la estructura base `GfxBase' podemos encontrar la dirección de memoria en cuestión. Para esto primero debemos abrir la librería *graphics* del sistema operativo, nos será regresada la dirección de memoria de la estructura `GfxBase' en un apuntador a la misma llamado *GfxBase. Después podemos usar los punteros a estructuras con el operador flecha de la siguiente forma:

```
GfxBase->ActiView->LOFCprList->start
```

para encontrar finalmente la dirección de memoria en donde se encuentra la lista o programa COPPER del despliegue actual o activo. Esto permitirá una comunicación basada en detectar la "pantalla señal generada por el programa ScalaMM.

6.4.2.2 El Programa principal

6.4.2.2.1 Alcance y planteamientos iniciales de programación

¿Que debe hacer nuestro programa? A continuación se listan los objetivos principales.

-Leer un archivo de texto con una especie de lenguaje o script, que indica cada una de las acciones a realizar.

-Establecer comunicación con programas externos ejecutados concurrentemente.

-Desplegar algunos efectos gráficos no presentes en el programa ScalaMM.

-Manejar los efectos gráficos de forma "asíncrona", es decir, no debe esperar a que termine el efecto o animación actual para continuar con otro.

Considerando lo anterior, nuestro programa principal simplemente leerá y ejecutará las acciones de un script. Esto simplifica el trabajo, pues si en un momento dado se desea crear una interfaz gráfica para facilitar el uso del mismo, no será necesario aumentar el tamaño de nuestro programa principal, pues se puede crear otro programa, inclusive en otro lenguaje, únicamente enfocado a generar los scripts.

La pregunta obligada es ¿En qué lenguaje será mejor realizar éste programa?, no hay mucho de donde elegir, sabemos que los compiladores de C y el lenguaje C mismo en esencia, permiten obtener el código más eficaz y rápido además de ser muy portable.

Sin embargo, sí podemos elegir entre C ó C++. C++ facilita mucho las cosas si el programa crece, siempre y cuando al principio se elija el camino más adecuado, y además cuenta con muchas clases gráficas ya realizadas listas para usarse, pero aquí nos preguntamos, ¿Las clases ya realizadas hacen el trabajo de la forma más rápida y óptima posible? y esto nos lleva al primer problema: velocidad. Es un hecho que un programa pequeño bien realizado en C estándar siempre será más rápido que su equivalente en C++. Además, si está bien planteado y documentado, es posible convertirlo a C++ cuando se requiera hacerlo más grande y complejo.

Más aún, nuestro programa principal no puede ser muy grande pues trabajará concurrentemente con otros y además debe realizar sus actividades lo más rápido posible y por lo tanto trabajar generalmente en bajo nivel. Como ya se mencionó, éste programa principal no requiere de interfaz gráfica y otros detalles relacionados que lo harían mas voluminoso, todo eso puede realizarse en otro programa y con un lenguaje orientado a objetos.

En este sentido, para el programa principal lo más adecuado es utilizar lenguaje C estándar o un lenguaje similar en cuanto a rendimiento.

· Lenguaje Basic alternativo

Hablamos de un lenguaje similar porque para esta arquitectura existe un lenguaje llamado Blitz Basic 2 que es muy parecido a C, maneja funciones, estructuras, apuntadores, etc. Y el compilador soporta macros y directivas de precompilación/compilación también similares a C. Es tan parecido que puede convertirse un código a lenguaje C y viceversa sin mayores problemas.

Ahora nos preguntamos ¿Porqué utilizar este lenguaje tipo Basic?, para contestar a continuación se listan las ventajas de éste lenguaje sobre un compilador de lenguaje C:

-Puede precompilar y hacer residentes todos los archivos a incluir de programación del sistema en lenguaje C, esto permite utilizar las estructuras del sistema y cualquier cosa relacionada con lenguaje C de una forma muy eficiente, pues no es necesario indicar archivos de cabecera y el tiempo de compilación es reducido en gran medida. Además las instrucciones de este lenguaje son tan similares que podemos programar como si lo estuviéramos haciendo en lenguaje C, solo que la sintaxis de este lenguaje es más clara y simple. Inclusive existen convertidores automatizados que pueden generar un programa totalmente funcional en C a partir de este tipo de código.

-El compilador de este lenguaje contiene también un ensamblador totalmente integrado al lenguaje, lo cual puede ser muy ventajoso si se sabe utilizar, pues todo el código se encuentra en un mismo lugar y es compilado/ensamblado de forma transparente.

-Además de todas las funciones, comandos y operadores similares o equivalentes del lenguaje C, cuenta con muchísimos comandos extras propios del lenguaje en librerías residentes que están realizadas en lenguaje ensamblador altamente optimizado, esto implica que es casi imposible obtener un equivalente en C que alcance la misma velocidad y rendimiento. Estas librerías implementan una amplia gama de funciones que hacen mucho más fácil y rápido el trabajo del programador: manejo de listas ligadas, manejo de objetos de tipo gráfico, programación simplificada de coprocesadores, manejo de recursos de audio, etc.

Conclusión: Simplemente por la reducción en el tiempo de compilación, y sabiendo que al final es *posible hacer la conversión a C, podemos considerar seriamente el uso de este lenguaje*, pero las librerías optimizadas de comandos extras o propios de este lenguaje deben tomarse en cuenta por el factor velocidad, pues algunas de estas librerías son mucho más rápidas que sus contrapartes en C, sin mencionar que muchas de estas no existen para lenguaje C en un formato sencillo y listo para usarse.

A continuación se muestran las características de programación más importantes de este lenguaje, así como sus correspondencias con el lenguaje 'C':

-Comentarios:

```
Lenguaje C: // ó /**/
BlitzBasic: ;
```

-Variables:

Tipo	Sufijo	Rango	Bytes
Byte	.b	+/-128	1
Word	.w	+/-32768	2
Long	.l	+/-2147483648	4
Quick	.q	+/-32768.0000	2
Float	.f	+/-9*10^18	4
String	.s		

```
Lenguaje C: int b=10; long int c=65555;      char d;
BlitzBasic:   b.b=10:        c.l=65555: DEFTYPE.b d
```

El manejo de cadenas es más sencillo y no pierde efectividad ó velocidad:

```
Lenguaje C: char cadena[]="test" ; strcat ...
BlitzBasic: cadena.s="test" ó cadena$="test" : cadena$=cadena$+"...
```

```
Lenguaje C: char cadena[10];
BlitzBasic: MaxLen cadena.s=10 ó MaxLen cadena$=10
```

-Constantes:

```
Lenguaje C: #define CONSTANTE 1000
BlitzBasic: #CONSTANTE=1000
```

-Control de flujo de programa y etiquetas

Contiene todos los comandos clásicos del Basic moderno y lenguajes estructurados: For-Step-Next, While-Wend, Repeat-Until, If-Else-Endif, Select-Case-EndSelect, etc. Pero además soporta las instrucciones goto y gosub que racionalmente utilizadas pueden incrementar la velocidad e inclusive la claridad del código. Para esto se utilizan las etiquetas, que como en C se definen con una palabra seguida del caracter dos puntos.

-Operadores:

Además del equivalente de los operadores clásicos de C: AND,OR,NOT,<>,=,LSL,ASL,etc. este lenguaje utiliza el caracter & (igual que C) para localizar la dirección de una variable y el caracter ? para localizar la dirección de una etiqueta del programa en memoria.

-Arreglos

```
Lenguaje C: long int arreglo[100][10][10];
BlitzBasic:          arreglo.l(100,10,10)
```

-Estructuras o nuevos tipos de datos:

```
Lenguaje C:

struct dir {
  char nombre[30];
  char ciudad[20];
  char estado[10];
  long int codigo;
};

struct emp {
  struct dir direccion;
  float salario;
};

struct emp empleado;
```

```
empleado.salario=3000.90;
empleado.direccion.codigo=14360;
```

BlitzBasic:

```
NEWTYPE .dir
  nombre.s
  ciudad.s
  estado.s
  codigo.l
End NEWTYPE

NEWTYPE .emp
  direccion.dir
  salario.f
End NEWTYPE

DEFTYPE .emp empleado

empleado\salario=3000.90
empleado\direccion\codigo=14360
```

-Apuntadores

Lenguaje C:

```
struct emp *apuntador;

apuntador=&empleado;
printf("Valor=%f",apuntador->salario);
```

BlitzBasic:

```
DEFTYPE .emp *apuntador

*apuntador=empleado
print "Valor=",*apuntador\salario
```

Se observa aquí que la notación para el uso de punteros es ligeramente distinta para hacer la lectura del código un poco más simple, sin embargo, no se pierde flexibilidad en comparación con el lenguaje 'C'.

-Funciones

Las funciones también están implementadas en este lenguaje, con características casi idénticas a las del lenguaje C. Cada vez que se llama a una función, ésta es copiada a un bloque de memoria que es liberado una vez que la función termina, esto implica que es posible la recursión y el uso de variables locales de manera similar al lenguaje C.

Lenguaje C:

```
#include <stdio.h>
void errorm(int a)
{
  if (a==1)
    {
    printf("Error num. 1\n");
    }
  if (a==2) printf("Error num. 2\n");
  return;
}

long fact(int n)
{
  int k;
  long a=1;
  for (k=2;k<=n;k++){
    a=a*k;
  }
  return a;
}

main(void)
{
printf("%d\n",fact(7));
}
```

BlitzBasic:

```
Statement errorm{a.w}
  if a=1
    nprint "Error num. 1"
  endif
  if a=2 then nprint "Error num. 2"
End Statement

Function fact{n.w}
  DEFTYPE.w k
  a.l=1
  For k=2 to n Step 1
    a=a*k
  Next
  Function Return a
End Function

nprint fact{7}
```

-Listas

Este lenguaje contiene una implementación de listas enlazadas sobre un arreglo de punteros, que se pueden utilizar con cualquier tipo de datos predefinido o definido por el usuario. Para esto simplemente se utiliza la siguiente instrucción:

```
Dim List empleados.emp(100)
```

La instrucción anterior crea un arreglo donde se permite un máximo de 100 elementos de la estructura 'emp' en la lista, el lenguaje mantiene un conteo de número de elementos en esta y un apuntador interno hacia el elemento actual. A cada estructura o elemento se le asigna un apuntador hacia el elemento siguiente y al anterior.

De este modo, cuando un elemento es insertado en la lista, se sigue el procedimiento clásico: cualquier localidad de memoria libre es usada para almacenar el NUEVO ELEMENTO, el APUNTADOR HACIA EL ELEMENTO SIGUIENTE asignado al ELEMENTO ACTUAL es copiado al APUNTADOR HACIA EL ELEMENTO SIGUIENTE asignado al NUEVO ELEMENTO y después, el mismo APUNTADOR HACIA EL ELEMENTO SIGUIENTE asignado al ELEMENTO ACTUAL, es cambiado para apuntar hacia el NUEVO ELEMENTO.

Después de definir este "arreglo de listas" podemos utilizar funciones como las siguientes, que regresan un valor falso (0) si se sobrepasa el límite o el número de elementos en la lista:

(P=Apuntador interno al elemento actual de esta lista)

```
ResetList empleados()     ;P es reinicializado o borrado (NULO).
AddItem( empleados() )    ;Agrega un elemento y P apunta a éste.
NextItem( empleados() )   ;P apunta al siguiente elemento.
KillItem( empleados() )   ;Elimina elemento actual y P apunta al anterior.
...
```

Sabemos todas las ventajas en cuanto a velocidad y ahorro de memoria al utilizar listas en lugar de arreglos, y si bien esto es perfectamente posible lenguaje 'C', generalmente la implementación estará realizada en C mismo, por el contrario, este lenguaje implementa las funciones de lista en ensamblador, con las ventajas que esto conlleva.

-Limitaciones con respecto al lenguaje 'C'

A pesar de todas sus ventajas, es importante mencionar que el compilador de este lenguaje Basic alternativo, tiene algunas pequeñas limitaciones con respecto a un buen compilador de C, pues aún no soporta algunas optimizaciones extra que permiten una ganancia en la velocidad de ejecución, como por ejemplo el uso de variables registro o modelos pequeños de compilación. Sin embargo, la velocidad en la generación y depuración de un programa 100% funcional así como las ventajas anteriormente descritas son suficientes incentivos como para considerarlo seriamente.

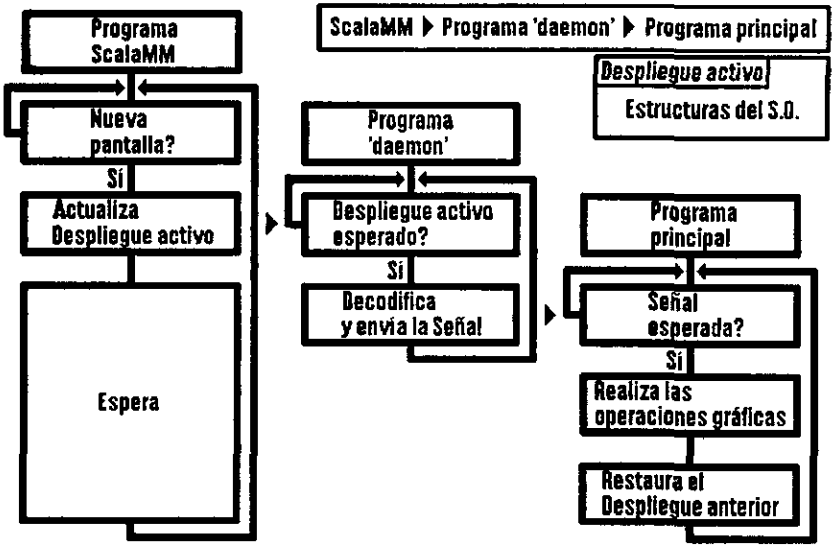
· Método final de solución

En este orden de ideas, nos conviene realizar el programa principal en el lenguaje BBasic2 para obener un código 100% funcional lo más rápido posible como ya se mecionó, posteriormente se podrà convertir a lenguaje C, si realmente es necesaria alguna ganancia adicional en velocidad.

El proceso de comunicación con el programa ScalaMM es repetitivo por naturaleza, pues es necesario comparar la lista de comandos Copper del despliegue gráfico activo o pantalla activa para saber si se trata de la "pantalla señal" requerida, además esto debe hacerse varias veces por segundo (60 veces por segundo, si es posible) para minimizar las posibles variaciones de tiempo que afectarían la sincronía del sistema multimedia en conjunto. En este sentido, el código para establecer la comunicación debe ser muy rápido y pequeño, y esto solo será posible utilizando lenguaje ensamblador o un buen compilador de C. La mejor opción es utilizar el compilador de C, pues a pesar de ser un programa más o menos simple, el compilador de C simplifica aún más las cosas y permite generar un código veloz, incluso para otros CPU's como el PPC, aunque esto implique una separación entre el código para establecer la comunicación y el programa principal.

El problema de la comunicación se reduce a realizar un programa externo que detecte la "pantalla señal" (o despliegue activo esperado) generada por el programa ScalaMM, para posteriormente enviar una señal a nuestro programa principal. Este programa externo debe ejecutarse concurrentemente con los otros dos (ScalaMM, Prog. Principal) y permanecer siempre activo, como un programa tipo 'daemon'.

Fig.24



6.4.2.2.2 El algoritmo principal de solución

· Efectos de video

La versión del hardware gráfico que estamos utilizando permite desplegar 2 pantallas independientes (despliegue dual) en paralelo y además puede desplegar 8 objetos gráficos independientes (*sprites*), que son de una anchura menor que una pantalla, pero usados conjuntamente nos permiten crear virtualmente una pantalla extra en paralelo.

Podemos basar nuestro programa en estos tres despliegues, que llamaremos:

backgr (despliegue de fondo)
foregr (despliegue frontal)
virtualgr (despliegue virtual)

Sobre estos tres elementos podemos definir algunas trayectorias o movimientos, por ahora solo en dos dimensiones, que serán la parte principal de los efectos de video. En este sentido, podemos almacenar y utilizar muchos movimientos predefinidos en el momento que así se requiera. Además existirán algunos efectos extras de video que podrán activarse paralelamente a estos movimientos.

· Script

Por otra parte, el usuario debe tener una forma de control para realizar lo siguiente:

-Seleccionar los archivos de imagen o animación sobre los que se va a trabajar.
-Activar los movimientos y efectos extras de video en el momento en que se requiera.

Para lograr esto, nuestro programa contará con un pequeño repertorio de comandos que podrán ser leídos desde un archivo de texto. Este archivo de texto contendrá un script que estará formado uno o más segmentos o partes del mismo, cada una de estas partes se ejecutarán de acuerdo a una señal proveniente del programa ScalaMM.

· Tiempo real y retardos.

Es importante mencionar que nuestro programa debe ejecutar todos los comandos del script en tiempo real, esto es, no habrá prácticamente ningún retardo entre cada instrucción, en este sentido, será responsabilidad del usuario el escribir un script con sus correspondientes retardos o comandos de espera.

Durante cada ciclo de "espera", serán realizadas, también en tiempo real, las acciones automatizadas correspondientes a la carga y decodificación de las nuevas imágenes requeridas, decodificación del siguiente cuadro de animación parcial de acuerdo al tiempo indicado, actualización de coordenadas de pantalla (movimientos) y efectos extras de video. Hay que identificar las dos últimas como acciones críticas, pues deben realizarse en tiempo real, pero además de acuerdo a la posición actual del barrido de pantalla para obtener una salida de video con buena calidad.

A continuación se muestra el diagrama de flujo que soluciona este problema, Utilizaremos las siglas: AT (Animación Técnica) como nombre clave para nuestro programa.

Fig.25

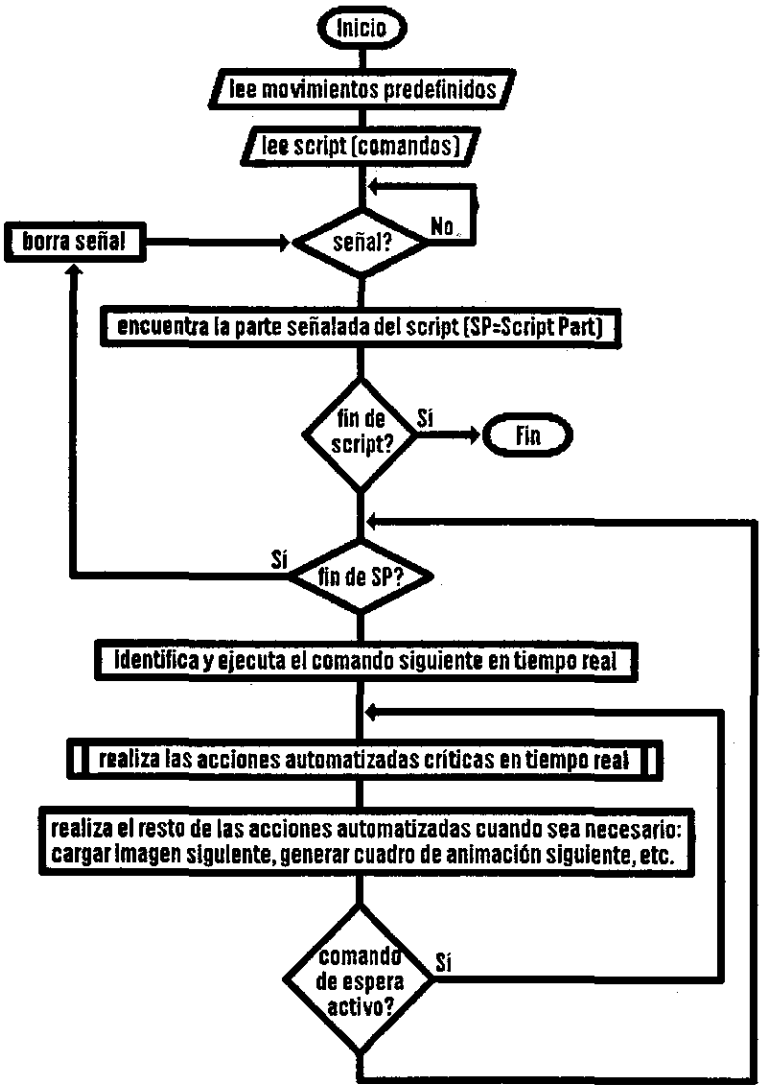


Diagrama de flujo del programa principal: ATPlayer

6.4.2.2.3 Listado final, script y explicaciones

Nota: Los listados finales están comentados en inglés para poder distribuirlos en Internet si se requiere.

```
;*************************************************
;**** A.T. V0.6 - 05/apr/99 By Carlos Villarroel ****
;*************************************************
WBStartup
;
.MainVars
; @
DEFTYPE .l atsignaldir  ; hold a signal mem. pos. to interact with mmplayer
DEFTYPE .l atstore      ; if no param. available atsignaldir points to this
DEFTYPE .s param        ; to process the startup parameters
DEFTYPE .l act,ini,fin  ; general purpose flags
DEFTYPE .l xact,yact    ; .
DEFTYPE .l xfin,yfin    ; .

; --- Next variables are also used in the IRQ code ---
DEFTYPE .l irqflag      ; to enable/disable the IRQ 5 (VBlank)
; --->Fade & XFade purposes:
DEFTYPE .q fade         ; Active Fade: 0=FadeOUT,1=FadeIN
DEFTYPE .l xfadenum     ; give the number of xfades to do & Start THE XFADE!
DEFTYPE .q fadespeed    ; +/-speed of the standar Fade (not xfade)
; --->PaintFill effect:
DEFTYPE .q paintfill    ; Active Paint Fill effect
DEFTYPE .q paintfillsp  ; +/-speed of the PFill effect
; --->Timers:
DEFTYPE .l counter      ; Custom timer (1/60 sec. NTSC)
DEFTYPE .l fdcounter    ; Custom timer for anim frame delay (1/60)
DEFTYPE .l movtimer     ; Time of the actual movement (1/60)
; --->Music volumes:
DEFTYPE .q musvol       ; Actual volume of music
DEFTYPE .l newmusvol    ; Requested volume of music
DEFTYPE .q musfadeval   ; +/-speed of volume adjust (Value/60Hz)

.Structures
; @ 2D movements
NEWTYPE .movement
; @ number of bmaps/sprites used for this movement
nids.l      ;number of objects ID's (more than 1 for super-fast anims*.)
; @ mov limits
right.l     ;area limit for movements, accordly to bitmap size, wrap purposes.
down.l      ;predefined movements must have these parameters already.
left.l
up.l
;
sprtch.l    ;flag for sprite propierties - 1st Channel To USE -
; @ init coords
xini.l      ;x & y initial coords. (0,0 in bitmaps, other in sprites)
yini.l      ;
; @ vel/acc
xv.q        ;x & y speed-direction vector (could be negative or zero)
yv.q
xa.q        ;x & y acceleration (could be negative or zero)
ya.q
;Reserved for a complex formula or table of mov.(vx,vy,ax,ay,id,time)
;currently, only lineal/parabolic w/acc vector movement.
; @ number of movs
idmovs.l    ;x-y movements for the actual ID before next ID - Sfast anim*
fmovs.l     ;movements for every frame loaded (for AUTOMATED load purposes)
nmovs.l     ;number of x-y movements before loop flag.
loops.l     ;number of loops
;
; --- WORKSPACE VARIABLES: ---
obj.l[20]   ;ID of object to use: .obj[0] and .obj[1] if double-buffer,
            ;                     more if super-fast animation.
```

```
loop.l      ;actual loop
nmov.l      ;actual number of x-y movements in the loop
idndx.l     ;actual object index for double-buffer or super-fast anims.
x.q         ;actual object position- use Qwrap(,,)
y.q
xvel.q      ;actual object speed
yvel.q
yodd.l      ;impar ypos, for laced purposes (sprites)
ymid.q      ;y position/2, for laced purposes (sprites)
idmov.l     ;actual number of ID x-y movements
fmov.l      ;actual number of movements in the frame loaded.
fready.l    ;frame ready flag (to load next)-when fmov=fmovs.
End NEWTYPE
;
DEFTYPE .movement *mv ;Pointer to the mov structure
;
; @ number of mov effects (100 elem. array for now)
frmem=100   ;Accordly with the free memory of the current system
; The (0) element of every mov must be: EMPTY or CLEAR
Dim backgr.movement(frmem)
Dim foregr.movement(frmem)
Dim sprt.movement(frmem*2) ; more of these than the others
;
NEWTYPE .sprites
first.l
last.l
End NEWTYPE
;
Dim virtgr.sprites(frmem) ; group of sprite movements
;
DEFTYPE .l backgrndx ; Currently used movement (index)
DEFTYPE .l foregrndx
DEFTYPE .l virtgrndx ; Currently used group of sprite movements

.StartupPars
; @ Mem. address as parameter
; Max len of long signed int: +-2147483648
If NumPars>0
  param$=Par$(1)
  While Len(param$)<10
    param$="0"+param$
  Wend
  ini=Val(Left$(param$,5))
  fin=Val(Right$(param$,5))
  atsignaldir=ini*100000+fin
Else ; If not parameters...
  ; **** PART TO EXECUTE OF THE SCRIPT ****
  atstore = 02 ; LOCATE THIS AS SIGNAL VALUE
  ; ****    ^^                          ****
  atsignaldir=&atstore
EndIf
;NPrint atsignaldir
;NPrint Peek.l(atsignaldir)

._P_R_O_C_S_
; @

CheckMemory:
;-------------------------------------------------------------------;
;                                                                   ;
; Function    : CheckMemory{}                                       ;
; Syntax      : CheckMemory{w.l,h.l,d.l}                            ;
;               ----------------------------------------------------;
;               w.l = Width of the bitmap                           ;
;               h.l = Height of the bitmap                          ;
;               d.l = Depth of the bitmap                           ;
;               ----------------------------------------------------;
; Purpose     : Returns TRUE if there's available mem for the       ;
```

```
;              required bitmap sizes. It uses bank 0 of AM Lib.      ;
;____________________________________________________________________;
Function CheckMemory{w.l,h.l,d.l}
DEFTYPE .l mem
SHARED irqflag
  mem=Reserve(0,8+(w*h*d)/8,%10) ; Check for ChipMem
  If mem=0    ; Not enough memory!
    irqflag=0
    AMIGA
    EZRequest "Out of memory!|bitmap of sizes:|"+Str$(w)+"x"+Str$(h)+"x"+Str$(d)
    NPrint "Out of memory!"
    Function Return -1
  EndIf
  Erase 0 ; Erase this bank for the next use
Function Return 1 ; If all goes good.
End Function

LoadDPF:
;____________________________________________________________________;
;                                                                    ;
; Statement  : LoadDPF{}                                             ;
; Syntax     : LoadDPF{n.l}                                          ;
;              ------------------------------------------------------;
;              n.l = Display to fill                                 ;
;                    1-BACKGround=#BACKGR (Palette 0)                ;
;                    2-FOREGround=#FOREGR (Palette 0)                ;
;              ------------------------------------------------------;
; Purpose    : Load special XDPF palettes (.___SUBRs___)             ;
;____________________________________________________________________;
Statement LoadXDPF{n.l}
If n=1 ;If mode 1
EndIf
End Statement

; @ image colour operations over hardware in real time (<(1/60)s)
.XFade
;____________________________________________________________________;
;                                                                    ;
; Statement  : XFadeInit{}                                           ;
; Syntax     : XFadeInit{Ptr.l,SrcPal,DstPal,SCol,Colors,Steps}      ;
;              ------------------------------------------------------;
;              Ptr.l  = Pointer to WorkArea                          ;
;              SrcPal = Source Palette (this palette is altered)     ;
;              DstPal = Destination Palette                          ;
;              SCol   = First Color of Palette Range                 ;
;              Colors = Number of Colors to Fade                     ;
;              Steps  = Number of steps to complete XFade            ;
;              ------------------------------------------------------;
; Purpose    : Initialize WorkArea for XFade handling                ;
;____________________________________________________________________;
Statement XFadeInit{Ptr.l,SrcPal,DstPal,SCol,Colors,Steps}
;---
  SrcPos.l = Addr Palette(SrcPal)      ; Extract
  DstPos.l = Addr Palette(DstPal)      ; RGB-Data
  SrcPos.l = Peek.l(SrcPos)+4          ; From
  DstPos.l = Peek.l(DstPos)+4          ; Palette-Structure

  Poke.w Ptr  ,SCol                    ; Store all
  Poke.w Ptr+2,Colors                  ; needed
  Poke.w Ptr+4,Steps                   ; values
  Poke.w Ptr+6,0                       ; into the
  Poke.l Ptr+8,SrcPos                  ; Fade-Structure

  GetReg a1,Ptr                        ; Tell the routine where to find the
                                       ;  structure
  GetReg a3,DstPos                     ; Destination Palette not in
                                       ;  structure
```

```
      MOVE.l 8(a1),a2                ; a2 = SrcPal a3 = DstPal
      MOVE.w (a1),d1                 ; d1 = startcolor
      MULS.w #12,d1                  ; multiply with 12 bytes
      ADD.l  d1,a2                   ; add to SrcPal
      ADD.l  d1,a3                   ; add to DstPal
      MOVE.w 2(a1),d5                ; d5 = Colors
      MULS.w #3,d5                   ; x 3 for R,G & B
      SUBQ.w #1,d5                   ; -1
      MOVE.w 4(a1),d4                ; d4 = Steps
      ADD.l  #$C,a1                  ; a1 points at color table

Loop: MOVEQ.l #0,d1                  ; clear the values
      MOVEQ.l #0,d2                  ; for 32-bit division later on
      MOVE.w (a3),d1                 ; d1 = dst color
      MOVE.w d1,4(a1)                ; store in table->dest
      MOVE.w (a2),d2                 ; d2 - src color
      MOVE.w d2,(a1)                 ; store in table->Col
      SUB.l  d2,d1                   ; Get the diff between src & dst
      BEQ    Skip                    ; Prevent a zero-divide
      DIVS   d4,d1                   ; Divide by steps
Skip: MOVE.w d1,2(a1)                ; store in table->Plus
      ADD.l  #6,a1                   ; a1 + 3 words
      ADD.l  #4,a2                   ; a2 + 1 long
      ADD.l  #4,a3                   ; a3 + 1 long
      DBRA   d5,Loop
;---
End Statement

;-------------------------------------------------------------------------;
;                                                                         ;
; Statement   : XFade{}                                                   ;
; Syntax      : XFade{Ptr.l,Pal}                                          ;
;               ----------------------------------------------------------;
;               Ptr.l  = Pointer to WorkArea (inited by XFadeInit)        ;
;               Pal    = Palette to be altered                            ;
;               ----------------------------------------------------------;
; Purpose     : Fade from one palette into the other                      ;
;-------------------------------------------------------------------------;
Statement XFade{Ptr.l,Pal}
;---
  GetReg a1,Ptr

      MOVE.l 8(a1),a2                ; a2 = SrcPal
      MOVE.w (a1),d1                 ; d1 = startcolor
      MULS.w #12,d1                  ; multiply with 12 bytes
      ADD.l  d1,a2                   ; add to SrcPal
      MOVE.w 2(a1),d5                ; d5 = Colors
      MULS.w #3,d5                   ; x 3 for R,G & B
      SUBQ.w #1,d5                   ; -1
      MOVE.w 4(a1),d4                ; d4 = Steps
      ADDQ   #1,6(a1)                ; increase counter
      MOVE.w 6(a1),d3                ; d3 = current step
      ADD.l  #$C,a1                  ; a1 points at color table
      EOR.w  d3,d4
      BNE Fadr

Fadc: MOVEQ  #0,d1                   ; Clear for long copy later
      MOVE.w 4(a1),d1                ; copy table->dest to table->Col
      MOVE.w d1,(a1)                 ;
      AND.w  #$FF00,d1               ; calculate from fraction
      SWAP   d1
      MOVE.l d1,(a2)+                ; copy to SrcPal
      ADD.l  #6,a1                   ; next entry
      DBRA   d5,Fadc                 ;
      BRA    Fadx                    ; End

Fadr: MOVEQ  #1,d1                   ; Clear for long copy later
      MOVE.w (a1),d1
```

```
      ADD.w  2(a1),d1                ; Col + Plus
      MOVE.w d1,(a1)
      AND.w  #$FF00,d1               ; calculate from fraction
      SWAP   d1
      MOVE.l d1,(a2)+
      ADD.l  #6,a1
      DBRA   d5,Fadr
Fadx:
;---
End Statement

;*****************************************************************************
.__M_A_I_N__
; @ Graphic objects related
CopyFile "ENV:Sys/Pointer.prefs","ENV:Sys/Pointer.bak" ; Backup system pointer
SpriteMode 2  ; 64 bit graphic-architecture/sprites
LoadSprites 0,0,"NULL.sp" ; Create or load the default NULL sprites.
LoadSprites 1,1,"NULL.sp" ; (even and odd)
;****** Initialize main values ******
XFWork.l=AllocMem (256*18,0) ; Work area for the ML XFade.
InitPalette 1,16  ; Background palette
InitPalette 2,16  ; Foreground palette
InitPalette 3,16  ; Extra Foreground palette
InitPalette 6,16  ; Palette for sprites
;
InitPalette 0,256 ; Destination and display palette!
InitPalette 4,256 ; Source Palette for the extended Dual Mode (xmode on)
InitPalette 5,48  ; Source Palette for the standard Dual Mode.
InitPalette 7,256 ; Extra palette for double buffer xfades.

.Constants
; @ Most used values
#SCRWIDTH =640  ; Screen size: for now, only NTSC limits.
#SCRHEIGHT=400
#BACKG=1        ; Anim render and palette indicators.
#FOREG=2
#XTRAF=3
#VIRTG=6
#BPLMOD1=$108   ; Hardware registers.
#BPLMOD2=$10A
#BPLCON0=$100
#XDPFSRCPAL=4   ; xmode src. palette
#DPFSRCPAL=5    ; standard mode src. palette

; @ Video effects related
xfadenum=0    ; No fade/xfade for now.
fade=-1       ;
paintfill=0   ; No paint-fill for now.

.MakeDisplay
; @ Create the copper list skeleton
InitCopList 0,44,200,$11938,8,256,-10 ; Max of 10 custom copper instructions
; DisplayAdjust 0,-2,8,0,16,0 ;underscan! (8. Adjust the modulos!)

; @ Custom modifications for the copper list skel
; Make the back playfield use colour registers 16->31
; by putting %0001110000000000 ($1c00) into BPLCON3
; and make the sprites use colour registers 32-47
; by putting %0000000000110011 ($0033) into BPLCON4
; also by putting %0000000001000000 ($0040) into BPLCON2
; we can invert the playfield priority and get...
; 1st 16 col. BackG & 2nd 16 col. ForeG->If DPF=Standar!
DisplayControls 0,$0040,$1c00,$0033 ;264 to BPLCON2
;DisplayControls 0,$0064,$1c00,$0033 ;25B to BPLCON2
;
; @ Custom copper program for raster-synced video effects
CopperReset 0,0,0
CopperMove #BPLCON0,Cvi(Mki$($8615)) ; STANDARD DPF
```

```
CopperMove #BPLMOD1,76     ; MOD accordly to size of default BitMap.
CopperMove #BPLMOD2,76     ;
CopperWait 0,43            ; Wait to BEGIN OF DISPLAY
CopperMove #BPLMOD1,76     ; MOD accordly to size of default...
CopperMove #BPLMOD2,76     ;
CopperWait 0,244           ; Wait to END OF DISPLAY
CopperMove #BPLMOD1,76     ; MOD accordly to size of default...
CopperMove #BPLMOD2,76     ;
CopperWait 0,244           ; Wait to END OF DISPLAY
CopperEnd                  ; End of copper program and end of display.
;
BLITZ:CreateDisplay 0          ; build display (coplist=0)
AMIGA                          ; restore old display
DisplayPalette 0,0             ; colors must be black first
;
.Install_IRQ
; @
irqflag=0         ; Keep the VBlank IRQ disabled when installed...
Gosub VBlank_IRQ ; Install the VBlank Interruption!

; *** Fill movement objects ***
; These must be repeated with other bitmaps for alternate management
; (mov1a,mov1b,mov2a,... OR even,odd) of the displays.

.MovObjs
; @ Predefined movement structures (can be read from disk)
;################ Transparent scroll Internet anim ##############################
USEPATH backgr(1)
\nids=1      ;one bitmap must be allocated
             ;not dbuffer nor super fast animations in background
\right=4*#SCRWIDTH -#SCRWIDTH
\down=#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1  ;It's not an sprite
\xini=0
\yini=0
\xv=1.5       ;velocity & direction
\yv=0
\xa=0        ;acceleration
\ya=0
\idmovs=0    ;no ID changes every n-movs.
\fmovs=0     ;NO! load next frame every #SCRWIDTH movs.
\nmovs=4*#SCRWIDTH -#SCRWIDTH ;number of x-y movements before loop flag.
\loops=4
USEPATH foregr(1)
\nids=2      ; two bitmaps must be allocated
\right=#SCRWIDTH -#SCRWIDTH
\down=#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1  ;It's not an sprite
\xini=0
\yini=0
\xv=0        ;No movements
\yv=0
\xa=0
\ya=0
\idmovs=0    ;no ID changes every n-movs, user controled idndx(*)
\fmovs=0     ;no load next frame flag.
\nmovs=0     ;no movement loop.**** CHECK FIRST!!
\loops=1
USEPATH sprt(1) ; This is invalid for the Internet anim....<<<<
; \obj[0]=    ;SPRITE ID's MUST BE PAIRS (0,2,4,...)
; \obj[1]=    ;             1,3,... ARE FOR INTERLACE PURPOSES
\nids=1       ; one sprite allocated
\right=0      ; It's a NULL sprite
\down=0
```

```
\left=0
\up=-400
\sprtch=0     ;Start at channel 0
\xini=0
\yini=0
\xv=0
\yv=-2
\xa=0
\ya=0
\idmovs=0
\fmovs=0
\nmovs=0      ; NO movements for now
\loops=0
;virtualground contains only sprt(1)
virtgr(1)\first=1:virtgr(1)\last=1
;################### Final credits scroll ###################################
USEPATH backgr(2)
\nids=1       ;one bitmap must be allocated
              ;not dbuffer nor super fast animations in background
\right=4*#SCRWIDTH -#SCRWIDTH
\down=#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1  ;It's not an sprite
\xini=0
\yini=0
\xv=1      ;velocity & direction
\yv=0
\xa=0        ;acceleration
\ya=0
\idmovs=0    ;no ID changes every n-movs.
\fmovs=#SCRWIDTH/1 ;load next frame every #SCRWIDTH/xv movs.
\nmovs=(4*#SCRWIDTH -#SCRWIDTH)/1 ;x-y movements before loop flag.
\loops=6
USEPATH foregr(2)
\nids=1       ; one bitmap, not db nor sfast anims
\right=4*#SCRWIDTH -#SCRWIDTH
\down=#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1  ;It's not an sprite
\xini=0
\yini=0
\xv=2      ;speed
\yv=0
\xa=0
\ya=0
\idmovs=0    ;no ID changes every n-movs, user controled idndx(*)
\fmovs=#SCRWIDTH/2  ;load next frame every #SCRWIDTH/xv movs.
\nmovs=(4*#SCRWIDTH -#SCRWIDTH)/2 ;x-y movs before loop flag.
\loops=12
USEPATH sprt(2)
; \obj[0]=    ;SPRITE ID's MUST BE PAIRS (0,2,4,...)
; \obj[1]=    ;             1,3,... ARE FOR INTERLACE PURPOSES
\nids=2       ; 2 sprites allocated for the automated db
\right=256+#SCRWIDTH  ;The size is 256x400 (obj[0] and obj[1])
\down=#SCRHEIGHT -#SCRHEIGHT ;
\left=-256
\up=-400                    ;
\sprtch=0    ;Start at channel 0
\xini=96
\yini=0
\xv=0
\yv=-.5
\xa=0
\ya=0
\idmovs=800 ; automated double buffer animation every n movs
\fmovs=800  ; load when the db changes objs.
```

```
\nmovs=800  ; number of movs
\loops=4
USEPATH sprt(3)
; \obj[0]=   ;SPRITE ID's MUST BE PAIRS (0,2,4,...)
; \obj[1]=   ;            1,3,... ARE FOR INTERLACE PURPOSES
\nids=2      ; 2 sprites allocated for the automated db
\right=256+#SCRWIDTH  ;The size is 256x400 (obj[0] and obj[1])
\down=#SCRHEIGHT -#SCRHEIGHT ;
\left=-256
\up=-400                    ;
\sprtch=4   ;Start at channel 4
\xini=96+256
\yini=0
\xv=0
\yv=-.5
\xa=0
\ya=0
\idmovs=800 ; automated double buffer animation every n movs
\fmovs=800  ; load when the db changes objs.
\nmovs=800  ; number of movs
\loops=4
; Virtual Ground contains:
virtgr(2)\first=2
virtgr(2)\last=3
;#################### Initial Introduction ###################################
USEPATH backgr(4)
\nids=2     ;two bitmaps must be allocated
            ;user controlled dbuffer in background
\right=#SCRWIDTH -#SCRWIDTH
\down=#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1  ;It's not an sprite
\xini=0
\yini=0
\xv=0
\yv=0       ;No speed and dir.
\xa=0       ;No acceleration
\ya=0
\idmovs=0   ;no ID changes every n-movs.
\fmovs=0    ;NO load next frame every #SCRHEIGHT/xv movs.
\nmovs=0    ;DON'T move it.
\loops=0
USEPATH foregr(4)
\nids=1     ; one bitmap, not db nor sfast anims
\right=8*#SCRWIDTH -#SCRWIDTH + 639 ; For extra NULL screen at end
\down=#SCRHEIGHT -#SCRHEIGHT  + 1   ; For extra NULL screen at end
\left=0
\up=0
\sprtch=-1  ;It's not an sprite
\xini=0
\yini=0
\xv=0     ;speed
\yv=0
\xa=.015    ;acc
\ya=0
\idmovs=0   ;NO ID changes every n-movs, user controled idndx(*)
\fmovs=0    ;NO load next frame every #SCRWIDTH/xv movs.
\nmovs=1156  ;7+extra NULL x-y movs/xv before loop flag.
\loops=1
USEPATH sprt(4)
; \obj[0]=   ;SPRITE ID's MUST BE PAIRS (0,2,4,...)
\nids=1      ; 1 sprite allocated
\right=256+192+#SCRWIDTH  ;The size is 256x400 (obj[0] and obj[1])
\down=#SCRHEIGHT -#SCRHEIGHT ;
\left=-256-192
\up=0                     ;
\sprtch=0   ;Start at channel 0
```

```
\xini=640
\yini=0
\xv=-3
\yv=0
\xa=.01
\ya=0
\idmovs=0    ; NO automated double buffer animation every n movs
\fmovs=0     ; NO load when the db changes objs.
\nmovs=430   ; number of movs
\loops=1
USEPATH sprt(5)
; \obj[0]=   ;SPRITE ID's MUST BE PAIRS (0,2,4,...)
\nids=1      ; 1 sprite allocated
\right=256+192+#SCRWIDTH  ;The size is 256x400 (obj[0] and obj[1])
\down=#SCRHEIGHT -#SCRHEIGHT ;
\left=-256-192
\up=0                     ;
\sprtch=4    ;Start at channel 4
\xini=640+256
\yini=0
\xv=-3
\yv=0
\xa=.01
\ya=0
\idmovs=0    ; NO automated double buffer animation every n movs
\fmovs=0     ; NO load when the db changes objs.
\nmovs=430   ; number of movs
\loops=1
; Virtual Ground contains:
virtgr(4)\first=4
virtgr(4)\last=5
;################### RIT/RED-UNAM presentation ########################
USEPATH backgr(6)
\nids=1      ;one bitmap must be allocated
             ;not dbuffer nor super fast animations in background
\right=#SCRWIDTH -#SCRWIDTH
\down=4*#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1   ;It's not an sprite
\xini=0
\yini=0 ;3*#SCRHEIGHT
\xv=0        ;velocity & direction
\yv=4
\xa=0        ;acceleration
\ya=0
\idmovs=0    ;no ID changes every n-movs.
\fmovs=0     ;NO load next frame every #SCRHEIGHT/xv movs.
\nmovs=(4*#SCRHEIGHT -#SCRHEIGHT)/4 ;movements (#SCRHEIGHT/xv) before loop flag.
\loops=8
USEPATH foregr(6)
\nids=1      ; one bitmap, not db nor sfast anims
\right=#SCRWIDTH -#SCRWIDTH
\down=4*#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1   ;It's not an sprite
\xini=0
\yini=0
\xv=0      ;speed
\yv=6
\xa=0
\ya=0
\idmovs=0    ;no ID changes every n-movs
\fmovs=#SCRHEIGHT/6 ;load next frame every #SCRHEIGHT/xv movs.
\nmovs=(4*#SCRHEIGHT -#SCRHEIGHT)/6 ;x-y movs before loop flag.
\loops=11
USEPATH sprt(6)
```

```
; \obj[0]=   ;SPRITE ID's MUST BE PAIRS (0,2,4,...)
; \obj[1]=   ;            1,3,... ARE FOR INTERLACE PURPOSES
\nids=2      ; 2 sprites allocated for the automated db
\right=256+#SCRWIDTH  ;The size is 256x400 (obj[0] and obj[1])
\down=#SCRHEIGHT -#SCRHEIGHT ;
\left=-256
\up=-400                    ;
\sprtch=0   ;Start at channel 0
\xini=96
\yini=0
\xv=0
\yv=-2
\xa=0
\ya=0
\idmovs=400/2 ; automated double buffer animation every n movs
\fmovs=0;400/2  ; load when the db changes objs.
\nmovs=400/2  ; number of movs
\loops=12
USEPATH sprt(7)
; \obj[0]=   ;SPRITE ID's MUST BE PAIRS (0,2,4,...)
; \obj[1]=   ;            1,3,... ARE FOR INTERLACE PURPOSES
\nids=2      ; 2 sprites allocated for the automated db
\right=256+#SCRWIDTH  ;The size is 256x400 (obj[0] and obj[1])
\down=#SCRHEIGHT -#SCRHEIGHT ;
\left=-256
\up=-400                    ;
\sprtch=4   ;Start at channel 4
\xini=96+256
\yini=0
\xv=0
\yv=-2
\xa=0
\ya=0
\idmovs=400/2 ; automated double buffer animation every n movs
\fmovs=0;400/2  ; load when the db changes objs.
\nmovs=400/2  ; number of movs
\loops=12
; Virtual Ground contains:
virtgr(6)\first=6
virtgr(6)\last=7
;################ NOC/NIC/TAC-AdminS/AtnU/Con presentation ####################
USEPATH backgr(8)
\nids=1     ;one bitmap must be allocated
            ;not dbuffer nor super fast animations in background
\right=2*#SCRWIDTH -#SCRWIDTH
\down=2*#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1  ;It's not an sprite
\xini=-3.2
\yini=-2
\xv=3.2     ;(640/nmovs)-velocity & direction
\yv=2
\xa=0       ;acceleration
\ya=0
\idmovs=0   ;no ID changes every n-movs.
\fmovs=0    ;NO load next frame every #SCRHEIGHT/xv movs.
\nmovs=0+(2*#SCRHEIGHT -#SCRHEIGHT)/2 ;movements (#SCRHEIGHT/xv) before loop flag.
\loops=20
USEPATH foregr(8)
\nids=1     ; one bitmap, not db nor sfast anims
\right=3*#SCRWIDTH -#SCRWIDTH
\down=#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1  ;It's not an sprite
\xini=0
\yini=0
```

```
\xv=4.792      ;speed
\yv=0
\xa=-.009
\ya=0
\idmovs=0    ;no ID changes every n-movs, user controled idndx(*)
\fmovs=0     ;load next frame every #SCRWIDTH/xv movs.
\nmovs=540   ;x-y movs before loop flag.
\loops=1
USEPATH sprt(8)
; \obj[0]=    ;SPRITE ID's MUST BE PAIRS (0,2,4,...)
; \obj[1]=    ;            1,3,... ARE FOR INTERLACE PURPOSES
\nids=2      ; 2 sprites allocated for the automated db
\right=256+#SCRWIDTH  ;The size is 256x400 (obj[0] and obj[1])
\down=#SCRHEIGHT -#SCRHEIGHT ;
\left=-256
\up=-400                    ;
\sprtch=0    ;Start at channel 0
\xini=0
\yini=-401
\xv=0
\yv=4
\xa=0
\ya=0
\idmovs=100; automated double buffer animation every n movs
\fmovs=100 ; load when the db changes objs.
\nmovs=100 ; number of movs
\loops=14
; Virtual Ground contains:
virtgr(8)\first=8
virtgr(8)\last=8
;################ DTD presentation ####################
USEPATH backgr(14)
\nids=1      ;one bitmap must be allocated
             ;not dbuffer nor super fast animations in background
\right=2*#SCRWIDTH -#SCRWIDTH
\down=2*#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1   ;It's not an sprite
\xini=-3.2
\yini=-2
\xv=3.2      ;(640/nmovs)-velocity & direction
\yv=2
\xa=0        ;acceleration
\ya=0
\idmovs=0    ;no ID changes every n-movs.
\fmovs=0     ;NO load next frame every #SCRHEIGHT/xv movs.
\nmovs=0+(2*#SCRHEIGHT -#SCRHEIGHT)/2 ;movements (#SCRHEIGHT/xv) before loop flag.
\loops=20
USEPATH foregr(14)
\nids=1      ; one bitmap, not db nor sfast anims
\right=#SCRWIDTH -#SCRWIDTH
\down=2+(3*#SCRHEIGHT -#SCRHEIGHT)
\left=0
\up=0
\sprtch=-1   ;It's not an sprite
\xini=-1     ;To center it a bit
\yini=0
\xv=0
\yv=2        ;speed
\xa=0
\ya=0
\idmovs=0    ;no ID changes every n-movs, user controled idndx(*)
\fmovs=0     ;load next frame every #SCRWIDTH/xv movs.
\nmovs=399   ;x-y movs before loop flag.
\loops=1
USEPATH sprt(14)
; \obj[0]=    ;SPRITE ID's MUST BE PAIRS (0,2,4,...)
```

```
; \obj[1]=   ;            1,3,... ARE FOR INTERLACE PURPOSES
\nids=2      ; 2 sprites allocated for the automated db
\right=256+#SCRWIDTH  ;The size is 256x400 (obj[0] and obj[1])
\down=#SCRHEIGHT -#SCRHEIGHT ;
\left=-256
\up=-400                    ;
\sprtch=0   ;Start at channel 0
\xini=0;0,224,449
\yini=-401
\xv=0
\yv=4
\xa=0
\ya=0
\idmovs=100 ; automated double buffer animation every n movs
\fmovs=100  ; load when the db changes objs.
\nmovs=100  ; number of movs
\loops=5
; Virtual Ground contains:
virtgr(14)\first=14
virtgr(14)\last=14
;
;################ WWW anim ##############################
USEPATH backgr(15)
\nids=2     ; two bitmaps must be allocated
\right=#SCRWIDTH -#SCRWIDTH
\down=#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1  ;It's not an sprite
\xini=0
\yini=0
\xv=0       ;No movements
\yv=0
\xa=0
\ya=0
\idmovs=0   ;no ID changes every n-movs, user controled idndx(*)
\fmovs=0    ;no load next frame flag.
\nmovs=0    ;no movement loop.**** CHECK FIRST!!
\loops=0
;
USEPATH foregr(15)
\nids=1     ;one bitmap must be allocated
            ;not dbuffer nor super fast animations in background
\right=4*#SCRWIDTH -#SCRWIDTH
\down=#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1  ;It's not an sprite
\xini=0
\yini=0
\xv=1.5       ;velocity & direction
\yv=0
\xa=0       ;acceleration
\ya=0
\idmovs=0   ;no ID changes every n-movs.
\fmovs=0    ;NO! load next frame every #SCRWIDTH movs.
\nmovs=4*#SCRWIDTH -#SCRWIDTH ;number of x-y movements before loop flag.
\loops=4
;################ EMail anim ####################
USEPATH backgr(16)
\nids=1     ;one bitmap must be allocated
            ;not dbuffer nor super fast animations in background
\right=2*#SCRWIDTH -#SCRWIDTH
\down=2*#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1  ;It's not an sprite
\xini=-3.2
```

```
\yini=-2
\xv=3.2      ;(640/nmovs)-velocity & direction
\yv=2
\xa=0        ;acceleration
\ya=0
\idmovs=0    ;no ID changes every n-movs.
\fmovs=0     ;NO load next frame every #SCRHEIGHT/xv movs.
\nmovs=0+(2*#SCRHEIGHT -#SCRHEIGHT)/2 ;movements (#SCRHEIGHT/xv) before loop flag.
\loops=20
USEPATH foregr(16)
\nids=1      ; one bitmap, not db nor sfast anims
\right=3*#SCRWIDTH -#SCRWIDTH
\down=#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1   ;It's not an sprite
\xini=0
\yini=0
\xv=2        ;speed
\yv=0
\xa=0
\ya=0
\idmovs=0    ;no ID changes every n-movs, user controled idndx(*)
\fmovs=0     ;load next frame every #SCRWIDTH/xv movs.
\nmovs=(2*#SCRWIDTH/2)-2 ;x-y movs before loop flag.
\loops=1
;###################### A500 animation ######################
USEPATH backgr(17)
\nids=1      ; one bitmap, not db nor sfast anims
\right=3*#SCRWIDTH -#SCRWIDTH
\down=#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1   ;It's not an sprite
\xini=0
\yini=0
\xv=2        ;speed
\yv=0
\xa=0
\ya=0
\idmovs=0    ;no ID changes every n-movs, user controled idndx(*)
\fmovs=0     ;load next frame every #SCRWIDTH/xv movs.
\nmovs=(2*#SCRWIDTH/2) ;x-y movs before loop flag.
\loops=20
USEPATH foregr(17)
\nids=2      ; two bitmaps must be allocated for this anim
\right=#SCRWIDTH -#SCRWIDTH
\down=#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1   ;It's not an sprite
\xini=0
\yini=0
\xv=0        ;No movements
\yv=0
\xa=0
\ya=0
\idmovs=0    ;no ID changes every n-movs, user controled idndx(*)
\fmovs=0     ;no load next frame flag.
\nmovs=0     ;no movement loop.**** CHECK FIRST!!
\loops=0
USEPATH sprt(17)
; \obj[0]=   ;SPRITE ID's MUST BE PAIRS (0,2,4,...)
; \obj[1]=   ;            1,3,... ARE FOR INTERLACE PURPOSES
\nids=2      ; 2 sprites allocated for the automated db
\right=256+#SCRWIDTH  ;The size is 256x400 (obj[0] and obj[1])
\down=#SCRHEIGHT -#SCRHEIGHT ;
\left=-256
```

```
\up=-400                        ;
\sprtch=0   ;Start at channel 0
\xini=0
\yini=-401
\xv=0
\yv=2
\xa=0
\ya=0
\idmovs=200 ; automated double buffer animation every n movs
\fmovs=200  ; load when the db changes objs.
\nmovs=200  ; number of movs
\loops=7
USEPATH sprt(18)
; \obj[0]=   ;SPRITE ID's MUST BE PAIRS (0,2,4,...)
; \obj[1]=   ;            1,3,... ARE FOR INTERLACE PURPOSES
\nids=2      ; 2 sprites allocated for the automated db
\right=256+#SCRWIDTH  ;The size is 256x400 (obj[0] and obj[1])
\down=#SCRHEIGHT -#SCRHEIGHT ;
\left=-256
\up=-400                        ;
\sprtch=4   ;Start at channel 4
\xini=256
\yini=-401
\xv=0
\yv=2
\xa=0
\ya=0
\idmovs=200 ; automated double buffer animation every n movs
\fmovs=200  ; load when the db changes objs.
\nmovs=200  ; number of movs
\loops=7
; Virtual Ground contains:
virtgr(17)\first=17
virtgr(17)\last=18
;###################### IRC animation ######################
USEPATH backgr(19)
\nids=1     ; one bitmap, not db nor sfast anims
\right=#SCRWIDTH -#SCRWIDTH
\down=#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1  ;It's not an sprite
\xini=0
\yini=0
\xv=0       ;No movements
\yv=0
\xa=0
\ya=0
\idmovs=0   ;no ID changes every n-movs.
\fmovs=0    ;no load next frame every #SCRWIDTH/xv movs.
\nmovs=0    ;no movs before loop flag.
\loops=0
USEPATH foregr(19)
\nids=2     ; two bitmaps must be allocated for this anim
\right=#SCRWIDTH -#SCRWIDTH
\down=#SCRHEIGHT -#SCRHEIGHT
\left=0
\up=0
\sprtch=-1  ;It's not an sprite
\xini=0
\yini=0
\xv=0       ;No movements
\yv=0
\xa=0
\ya=0
\idmovs=0   ;no ID changes every n-movs, user controled idndx(*)
\fmovs=0    ;no load next frame flag.
\nmovs=0    ;no movement loop.
```

```
\loops=0
;
;
.Script
; @ Fast access script (no index) of unfixed size. (even strings)
NEWTYPE .command
name.s
nparam.l
sparam.s
End NEWTYPE
Dim List script.command(2000)

; ****************** READ SCRIPT FROM DISK *******************
If ReadFile(0,"AT_SCRIPT1.TXT")
  FileInput 0
  While NOT Eof(0)
    a$=Inkey$
    If a$=";"                          ; It's a comment?
      While Inkey$<>Chr$(10):Wend      ; Don't use <;> without <ENTER>!
    EndIf
    If Asc(a$)>64 AND Asc(a$)<96       ; It's a command?
      name$=""
      While a$<>";" AND a$<>Chr$(10)   ; Read line until <;> or <ENTER>
        If a$="+"                      ; It's a concatenatation symbol <+>?
          a$=Inkey$                    ; then read the <ENTER>,
          a$=Inkey$                    ; take the next char to join a$
        EndIf                          ; and continue normally...
        name$=name$+a$
        a$=Inkey$
      Wend
      AddItem script()                 ; Build structures
      fin=Instr (name$," ",0)
      If fin
        script()\name=Left$(name$,fin-1)
        name$=UnRight$(name$,fin)
      Else
        script()\name=name$
      EndIf
      fin=Instr (name$," ",0)
      If fin
        script()\nparam=Val(Left$(name$,fin-1))
        script()\sparam=UnRight$(name$,fin)
      Else
        script()\nparam=Val(name$)
      EndIf
    EndIf
  Wend
EndIf

.___BEGIN___
; @ ****************** BEGIN OF MAIN PART *******************
;
; Important related variables:
; backgr(),foregr(),virtgr(),sprt() - Movements
; backgrndx - Currently used movement (index)
; foregrndx -      "
; virtgrndx -      "          group of movs.

; @ two stacks (bmaps&sprites) to avoid using repeated id's!
Dim List freebitmaps.l(20)   ; Max number of bitmap objects
For ini=1 To 12              ; Bitmap 0 object is reserved!
  AddItem freebitmaps()      ; add id's
  freebitmaps()=ini
Next
Dim List freesprites.l(30)   ; Max number of sprite objects
For ini=2 To 24 Step 2       ; sprite 0-1 objects are reserved!
  AddItem freesprites()      ; add id's
  freesprites()=ini
```

```
Next

; @ Related vars.
DEFTYPE.q scripttime     ; time of the last played script part (MMPlay purposes)
DEFTYPE.l srcpalette     ; source palette of the actual dual std-mode/xmode
DEFTYPE.l db             ; double-buffer anim flag
DEFTYPE.l spritespri     ; flag of sprites priority: 0=front 1=back 2=mid
DEFTYPE.l animop7flag    ; flag to enable AnimOP7 properties
DEFTYPE.l actmus         ; (-1)=Loaded music in mem. (0)=No music in mem.
DEFTYPE.l autofgpalfade ; fades between palettes for the automated ForeG frames
DEFTYPE.l playmain       ; main animation status: 0=OFF, 1=ON
DEFTYPE.l rendmain       ; main animation render target: #BACKG or #FOREG
                         ; and destination palette.
DEFTYPE.l waitflag       ; if <wait main frame> or <wait> commands used.
DEFTYPE.l waittime       ; time to wait (1/60 sec. NTSC)
DEFTYPE.l waitframe      ; frame of main animation to wait
DEFTYPE.l waitmov        ; time to wait from the begin of the last movement
DEFTYPE.l frame          ; actual frame of main animation
DEFTYPE.l framedelay     ; delay between frames
; @ new_ used to install new movements while others are present...
; @ to avoid delays between current and next. (sort of D.Buffer.)
DEFTYPE.l new_backgrndx ; next movement to use (index)
DEFTYPE.l new_foregrndx ; "
DEFTYPE.l new_virtgrndx ; next group of movs. to use
DEFTYPE.l new_xtrafg     ; extra effect loaded
DEFTYPE.l bgframec       ; frame counter (1-4)
DEFTYPE.l fgframec       ; for the automated loader
DEFTYPE.l vgframec       ; ...
DEFTYPE.s name           ; actual name to load
;
DEFTYPE.l bgnameini      ; range of names to load
DEFTYPE.l bgnamefin
DEFTYPE.l fgnameini
DEFTYPE.l fgnamefin
DEFTYPE.l vgnameini
DEFTYPE.l vgnamefin
DEFTYPE.l new_bgnameini ; range of new names to load
DEFTYPE.l new_bgnamefin
DEFTYPE.l new_fgnameini
DEFTYPE.l new_fgnamefin
DEFTYPE.l new_vgnameini
DEFTYPE.l new_vgnamefin
DEFTYPE.s bgframe        ; names for the automated loader
DEFTYPE.s fgframe
DEFTYPE.s vgframe
DEFTYPE.s new_bgframe    ; names for the new automated loader
DEFTYPE.s new_fgframe
DEFTYPE.s new_vgframe
;
AnimLoop On              ; the animations will loop around...
SetPTCia                 ; correct music speed regardless of the screen mode
SetPTVolume 64           ; max. volume at start
musvol=64:newmusvol=64   ; "
SetPTSongLoop On         ; the music will loop around...
TimerReset ; Reset default timer in the very begining...                    ; -TIMER#####
;
.MainLoop:
; @
Repeat ; Repeat main loop FOREVER
;
scripttime=Timer/60
NPrint "The time in seconds of the last part played is: ",scripttime ; -TIMER#####
irqflag=0 ; Force disable irq (not needed if the script is well written).
;
; @ Wait for a new signal... (VWait to save CPU)
While Peek.l(atsignaldir) = $FFE
  VWait
Wend
```

```
ResetTimer ; timer to zero every new play of the script part          ; -TIMER#####
backgrndx=0 ; Default null movement objects from the begin.
foregrndx=0 ;
virtgrndx=0 ;
InitPalette #XDPFSRCPAL,256 ; And a black display palettes!
InitPalette #DPFSRCPAL,48   ;
InitPalette 0,256           ;
;
; @ Find part of the script to execute, accordly to atsignal.
ResetList script()
Repeat
  act=NextItem(script())
Until act=0 OR ( script()\name="PART" AND script()\nparam=Peek.l(atsignaldir) )
; Until it's the signal value or...
; @ <**** END_OF_PROGRAM ****> if NO script part found.
If act=0
  .EOP:
  Poke.l(atsignaldir),$FFF ; Send the END_OF_PROGRAM signal to the ATdaemon.
  ClrInt 5                 ; Clear the VBlank IRQ
  CopyFile "ENV:Sys/Pointer.bak","ENV:Sys/Pointer.prefs" ; Restore system pointer.
  KillFile "ENV:Sys/Pointer.bak"
  End       ; And free the rest of memory/objects to return to the system!
EndIf
; @ START THE SCRIPT PART LOOP!
act=NextItem(script())
While act       ; Until it's the END command or the very last command.
  Select script()\name
  ;
  ; ********** fast commands/instructions/events **********
  Case "ENABLE_XMODE"
    xfadenum=0 ; Disable X-Fades before!
    CopperReset 0,0,0                    ; SPECIAL DPF MODE PROGRAMMED BY ME
    VWait ; This to avoid a little flicker...
    CopperMove #BPLCON0,Cvi(Mki$($8215)) ; BPLCON0 byte-4
    DisplayPalette 0,#XDPFSRCPAL
    srcpalette=#XDPFSRCPAL
    DuplicatePalette #XDPFSRCPAL,0
    ;VWait ; This to avoid a little flicker...
    DisplayPalette 0,0
  ;
  Case "DISABLE_XMODE"
    xfadenum=0 ; Disable X-Fades before!
    CopperReset 0,0,0
    VWait ; This to avoid a little flicker...
    CopperMove #BPLCON0,Cvi(Mki$($8615)) ; STANDARD DPF
    DisplayPalette 0,#DPFSRCPAL
    srcpalette=#DPFSRCPAL
    DuplicatePalette #DPFSRCPAL,0
    ;VWait ; This to avoid a little flicker...
    DisplayPalette 0,0
  ;
  Case "ACTIVE_MOVS" ; Active, and free the last!
    irqflag=0 ; Disable for a while
    fin=1 ; Flag to disable irq and load null sprites if new movs. are null.
    If new_backgrndx
      ini=0                              ; - Free the old bitmaps
      While backgr(backgrndx)\obj[ini] ; If movement\obj[]<>NULL
        AddItem freebitmaps()
        freebitmaps()=backgr(backgrndx)\obj[ini]
        Free BitMap freebitmaps()
        backgr(backgrndx)\obj[ini]=0    ; Clear them from the object
        ini+1
      Wend
      If new_backgrndx<>-1
        fin=0
        backgrndx=new_backgrndx:bgnameini=new_bgnameini:bgframe$=new_bgframe$
        bgframec=4 ; If exists, the new automated loader begins at frame 4
        ;
```

```
    *mv=backgr(backgrndx):Gosub InitMov
    act=(( #SCRWIDTH*Int( (*mv\right+#SCRWIDTH)/#SCRWIDTH ) -320)/4)-4 ; modulo
    ;act=((                *mv\right+#SCRWIDTH              -320)/4)-4 ; formula
    CopperReset 0,0,1
    CopperMove #BPLMOD1,act
    CopperReset 0,0,4
    CopperMove #BPLMOD1,act
    CopperReset 0,0,7
    CopperMove #BPLMOD1,act
  EndIf
EndIf
;
If new_foregrndx
  ini=0                              ; - Free the old bitmaps
  While foregr(foregrndx)\obj[ini] ; If movement\obj[]<>NULL
    AddItem freebitmaps()
    freebitmaps()=foregr(foregrndx)\obj[ini]
    Free BitMap freebitmaps()
    foregr(backgrndx)\obj[ini]=0   ; Clear them from the object
    ini+1
  Wend
  If new_foregrndx<>-1
    fin=0
    foregrndx=new_foregrndx:fgnameini=new_fgnameini:fgframe$=new_fgframe$
    fgframec=4 ; If exists, the new automated loader begins at frame 4
    ;
    *mv=foregr(foregrndx):Gosub InitMov
    If new_xtrafg  ; Show the correct bitmap if extraforeground effect!
      *mv\idndx=3
    EndIf
    act=(( #SCRWIDTH*Int( (*mv\right+#SCRWIDTH)/#SCRWIDTH ) -320)/4)-4 ; modulo
    ;act=((                *mv\right+#SCRWIDTH              -320)/4)-4 ; formula
    CopperReset 0,0,2
    CopperMove #BPLMOD2,act
    CopperReset 0,0,5
    CopperMove #BPLMOD2,act
    CopperReset 0,0,8
    CopperMove #BPLMOD2,act
  EndIf
EndIf
;
If new_virtgrndx ; (*) Or put NULL sprites before next mov?.
  For act=virtgr(virtgrndx)\first To virtgr(virtgrndx)\last
    ini=0                              ; - Free the old sprites
    While sprt(act)\obj[ini] ; If movement\obj[]<>NULL
      AddItem freesprites()
      freesprites()=sprt(act)\obj[ini]
      Free Sprite freesprites()    ; Free the even sprite...
      Free Sprite freesprites()+1 ; ...and the odd sprite (interlaced)
      sprt(act)\obj[ini]=0         ; Clear them from the object
      ini+1
    Wend
  Next
  If new_virtgrndx<>-1
    fin=0
    virtgrndx=new_virtgrndx:vgnameini=new_vgnameini:vgframe$=new_vgframe$
    ;
    For act=virtgr(virtgrndx)\first To virtgr(virtgrndx)\last
      *mv=sprt(act):Gosub InitMov
    Next
  Else ; NULL virtualground? then put NULL (sprite 0) sprites...(*)
    For act=0 To 7 : DisplaySprite 0,0,0,0,act : Next
  EndIf
EndIf
;
new_backgrndx=0:new_foregrndx=0:new_virtgrndx=0:new_xtrafg=0
movtimer=0                ; The movement timer count begins...
If fin=0 Then irqflag=1 ; Not end of script part? Enable now the new movs!
```

```
  ;
  animop7flag=0           ; By default no animop7 properties.
  autofgpalfade=0         ; By default no automated foreground pal. fades.
  ;
  Goto end_InitMov:
  ;
  .InitMov:
    *mv\loop=*mv\loops ; initialize mov. obj. workspace
    *mv\nmov=0
    *mv\idndx=0
    *mv\x=*mv\xini
    *mv\y=*mv\yini
    *mv\xvel=*mv\xv
    *mv\yvel=*mv\yv
    *mv\idmov=0
    *mv\fmov=0
    *mv\fready=0
    Return
  end_InitMov:
;
Case "WAIT"
  waittime=script()\nparam
  counter=0
  waitflag=1
;
Case "WAIT_MAIN_FRAME"
  waitframe=script()\nparam
  waitflag=1
;
Case "WAIT_FROM_LAST_MOV"
  waitmov=script()\nparam
  waitflag=1
;
Case "GET_BACKGROUND"  ; ******** For the normal effects...
  For ini=0 To 15:CopyColour #BACKG,#DPFSRCPAL,ini,ini:Next ini
;
Case "GET_FOREGROUND"
  For ini=0 To 15:CopyColour #FOREG,#DPFSRCPAL,ini,ini+16:Next ini
;
Case "GET_SPRITES"
  For ini=0 To 15:CopyColour #VIRTG,#DPFSRCPAL,ini,ini+32:Next ini
;
Case "GET_BLANK"
  InitPalette #DPFSRCPAL,48
;
Case "FADE_SPEED"
  XFadeInit{XFWork,0,#DPFSRCPAL,1,47,script()\nparam}
;
Case "SHOW_FADE"
  xfadenum=script()\nparam ;Start the standard fade!
;
Case "GET_XBACKGROUND" ; ******** For the x-mode effects...
  Gosub LoadDPFpalB ; Initial palette 4 #XDPFSRCPAL
;
Case "GET_XFOREGROUND"
  Gosub LoadDPFpalF ; Initial palette 4 #XDPFSRCPAL
;
Case "GET_XBACKOVERFOREGROUND"
  Gosub LoadDPFpalB
  endreg=1:Gosub LoadDPFpalF
;
Case "GET_XFOREOVERBACKGROUND"
  Gosub LoadDPFpalF
  endreg=1:Gosub LoadDPFpalB
;
Case "GET_XEXTRAFOREGROUND"
  DuplicatePalette 0,#XDPFSRCPAL
  customdepth=1
```

```
  Gosub LoadDPFpalF
  endreg=1:Gosub LoadDPFpalB
;
Case "XFADE_SPEED" ; param=steps
  XFadeInit{XFWork,0,#XDPFSRCPAL,1,255,script()\nparam}
;
Case "XFADE_IN_SPEED"
  DuplicatePalette 0,7
  XFadeInit{XFWork,0,#XDPFSRCPAL,1,255,script()\nparam}
;
Case "XFADE_OUT_SPEED"
  XFadeInit{XFWork,0,7,1,255,script()\nparam}
;
Case "SHOW_XFADE"
  xfadenum=script()\nparam ;Start the xfade!
;
Case "SHOW_GFADE_IN" ; param=steps
  fadespeed=.00392*(255/script()\nparam):fade=0  ; Do the Fade!
;
Case "SHOW_GFADE_OUT"
  DuplicatePalette 0,#XDPFSRCPAL
  fadespeed=-.00392*(255/script()\nparam):fade=1 ; Do the Fade!
;
Case "SHOW_PAINTFILL"
  If script()\sparam="background"
    CopperReset 0,0,7
    CopperMove #BPLMOD1,-76-8
    CopperMove #BPLMOD1,-76-8
  EndIf
  If script()\sparam="foreground"
    CopperReset 0,0,7
    CopperMove #BPLMOD2,-76-8
    CopperMove #BPLMOD2,-76-8
  EndIf
  If script()\sparam=""
    CopperReset 0,0,7
    CopperMove #BPLMOD1,-76-8
    CopperMove #BPLMOD2,-76-8
  EndIf
  paintfill=1
  paintfillsp=script()\nparam
;
Case "SHOW_INV_PAINTFILL"
  If script()\sparam="background"
    CopperReset 0,0,4
    CopperMove #BPLMOD1,-76-8
    CopperMove #BPLMOD1,-76-8
  EndIf
  If script()\sparam="foreground"
    CopperReset 0,0,4
    CopperMove #BPLMOD2,-76-8
    CopperMove #BPLMOD2,-76-8
  EndIf
  If script()\sparam=""
    CopperReset 0,0,4
    CopperMove #BPLMOD1,-76-8
    CopperMove #BPLMOD2,-76-8
  EndIf
  paintfill=200
  paintfillsp=-script()\nparam
;
Case "SHOW"
  xfadenum=0 ; Disable X-Fades before!
  DisplayPalette 0,srcpalette
  DuplicatePalette srcpalette,0
  DisplayPalette 0,0
;
; ************** allocate & load events ***************
```

```
  Case "LOAD_NULL_MOVS"
    new_backgrndx=-1
    new_foregrndx=-1
    new_virtgrndx=-1
  ;
  Case "LOAD_EXTRAFOREGROUND_MOV" ; - This is a special mov:
    new_foregrndx=script()\nparam
    new_fgframe$=UnLeft$(script()\sparam,2)
    new_xtrafg=1
    *mv=foregr(new_foregrndx)
          If CheckMemory{*mv\right+#SCRWIDTH,*mv\down+#SCRHEIGHT,1}=-1 Then Goto EOP
    BitMap freebitmaps(),*mv\right+#SCRWIDTH,*mv\down+#SCRHEIGHT,1
    *mv\obj[0]=freebitmaps()
    KillItem freebitmaps()
          If CheckMemory{*mv\right+#SCRWIDTH,*mv\down+#SCRHEIGHT,1}=-1 Then Goto EOP
    BitMap freebitmaps(),*mv\right+#SCRWIDTH,*mv\down+#SCRHEIGHT,1
    *mv\obj[1]=freebitmaps()
    KillItem freebitmaps()
          If CheckMemory{*mv\right+#SCRWIDTH,*mv\down+#SCRHEIGHT,3}=-1 Then Goto EOP
    BitMap freebitmaps(),*mv\right+#SCRWIDTH,*mv\down+#SCRHEIGHT,3
    *mv\obj[2]=freebitmaps()
    If BLoad (new_fgframe$,0)       ; Bank 0 is for images only!!
      UnpackIFF Start(0),freebitmaps()
      ILBMPalette Start(0),#FOREG,8    ; Fill foregr pal last 8 regs.
      ; Repeat the last 8 colours for the effect.
      For ini=0 To 7:CopyColour 2,2,8+ini,0+ini:Next ini
    EndIf
    KillItem freebitmaps()
    MergeBitplanes *mv\obj[2],*mv\obj[0],freebitmaps(),4
    *mv\obj[3]=freebitmaps()
    KillItem freebitmaps()
    MergeBitplanes *mv\obj[2],*mv\obj[1],freebitmaps(),4
    *mv\obj[4]=freebitmaps()
    KillItem freebitmaps()
    ; *mv\nids=2 (2 bitmaps needed for dbuffer) this time not used.
    ; Then *mv\idndx=3 to show the virtual bitmap!
  ;
  Case "LOAD_BACKGROUND_MOV"
    new_backgrndx=script()\nparam
    new_bgframe$=script()\sparam
    *mv=backgr(new_backgrndx)
    If *mv\nids           ; If the mov. need bitplanes (or it's blank)
      new_bgnameini=1     ; There's the normal mov:
      For ini=0 To *mv\nids-1    ; >>> Assign number of bitmaps needed (nids)
        xfin=(*mv\right+#SCRWIDTH)/#SCRWIDTH      ; The x value must be /#SCRWIDTH
            If CheckMemory{xfin*#SCRWIDTH,*mv\down+#SCRHEIGHT,4}=-1 Then Goto EOP
        BitMap freebitmaps(),xfin*#SCRWIDTH,*mv\down+#SCRHEIGHT,4
        *mv\obj[ini]=freebitmaps()
        yfin=(*mv\down+#SCRHEIGHT)/#SCRHEIGHT     ; Now LOAD the bitmaps
        If *mv\fmovs<>0 AND xfin=4 Then xfin=3 ; It's an automated movement
        If *mv\fmovs<>0 AND yfin=4 Then yfin=3 ; It's an automated movement
        For xact=1 To xfin
          For yact=1 To yfin
            new_bgnamefin=Instr (new_bgframe$,",",new_bgnameini)
            If new_bgnamefin                    ; There's another name to load...
              name$=Mid$(new_bgframe$,new_bgnameini,new_bgnamefin-new_bgnameini)
              new_bgnameini=new_bgnamefin+1     ;
    If BLoad (name$,0)                          ; Bank 0 is for images only!!
    act=0:If xfin>1 AND yfin>1 Then act=yact-1 ; If the width and height are big
UnpackIFF Start(0),freebitmaps(),#SCRHEIGHT,(xact-1)*(#SCRWIDTH/8)+(yact-
1+act)*(#SCRWIDTH/8)*#SCRHEIGHT
    EndIf
            EndIf                               ;
          Next
        Next
        KillItem freebitmaps()
      Next                        ; >>>
      ILBMPalette Start(0),#BACKG ; background palette
```

```
    EndIf                ;
  ;
  Case "LOAD_FOREGROUND_MOV"
    new_foregrndx=script()\nparam
    new_fgframe$=script()\sparam
    *mv=foregr(new_foregrndx)
    If *mv\nids          ; If the mov. need bitplanes (or it's blank)
      new_fgnameini=1    ; There's the normal mov:
      For ini=0 To *mv\nids-1   ; >>> Assign number of bitmaps needed (nids)
        xfin=(*mv\right+#SCRWIDTH)/#SCRWIDTH      ; The x-value must be /#SCRWIDTH
            If CheckMemory(xfin*#SCRWIDTH,*mv\down+#SCRHEIGHT,4)=-1 Then Goto EOP
        BitMap freebitmaps(),xfin*#SCRWIDTH,*mv\down+#SCRHEIGHT,4
        *mv\obj[ini]=freebitmaps()
        yfin=(*mv\down+#SCRHEIGHT)/#SCRHEIGHT     ; Now LOAD the bitmaps
        If *mv\fmovs<>0 AND xfin=4 Then xfin=3 ; It's an automated movement
        If *mv\fmovs<>0 AND yfin=4 Then yfin=3 ; It's an automated movement
        For xact=1 To xfin
          For yact=1 To yfin
            new_fgnamefin=Instr (new_fgframe$,",",new_fgnameini)
            If new_fgnamefin                    ; There's another name to load...
              name$=Mid$(new_fgframe$,new_fgnameini,new_fgnamefin-new_fgnameini)
              new_fgnameini=new_fgnamefin+1     ;
    If BLoad (name$,0)                          ; Bank 0 is for images only!!
    UnpackIFF Start(0),freebitmaps(),#SCRHEIGHT,(xact-1)*(#SCRWIDTH/8)+(yact-
1)*(#SCRWIDTH/8)*#SCRHEIGHT
    EndIf
          EndIf                                 ;
        Next
      Next
      KillItem freebitmaps()
    Next                        ; >>>
    ILBMPalette Start(0),#FOREG ; foreground palette
  EndIf                ;
  ;
  Case "LOAD_VIRTUALGROUND_MOV"
    new_virtgrndx=script()\nparam
    new_vgframe$=script()\sparam
    new_vgnameini=1
    For act=virtgr(new_virtgrndx)\first To virtgr(new_virtgrndx)\last
      *mv=sprt(act)
      If *mv\nids          ; If the mov. need sprites (or it's blank)
        For ini=0 To *mv\nids-1   ; >>> Assign number of sprites needed (nids)
          *mv\obj[ini]=freesprites()
          new_vgnamefin=Instr (new_vgframe$,",",new_vgnameini)
          If new_vgnamefin                      ; There's another name to load...
            name$=Mid$(new_vgframe$,new_vgnameini,new_vgnamefin-new_vgnameini)
            new_vgnameini=new_vgnamefin+1       ;
            LoadSprites freesprites(),freesprites(),name$ ; LOAD it!
            LoadSprites freesprites()+1,freesprites()+1,name$+"o"
            ; IMPORTANT-The interlaced sprites must be saved:even(0)+odd(1)
          EndIf                                 ;
          KillItem freesprites()
        Next                        ; >>>
        ;TEST PURPOSES:
        ;DisplaySprite 0,*mv\obj[0],0,0,0
        ;DisplaySprite 0,*mv\obj[0]+1,300,0,4
        ;DisplaySprite 0,*mv\obj[1],0,0,0
        ;DisplaySprite 0,*mv\obj[1]+1,300,0,4
      EndIf                ;
    Next
    new_vgnamefin=Instr (new_vgframe$,",",new_vgnameini) ;Now load palette
    If new_vgnamefin         ; The last name exists?
      name$=Mid$(new_vgframe$,new_vgnameini,new_vgnamefin-new_vgnameini)
      new_vgnameini=new_vgnamefin+1
      LoadPalette #VIRTG,name$    ; Next name is palette for virtualground
    EndIf
  ;
  Case "LOAD_ANIM"
```

```
  If script()\nparam>0 Then BLoad script()\sparam,script()\nparam ; Bank>0!
;
Case "PLAY_MAIN_ANIM"
  If script()\sparam="background"
    *mv=backgr(backgrndx)
    If *mv\obj[1] ; If there's room for double buffer (2 bmaps)
      rendmain=#BACKG
    EndIf
  EndIf
  If script()\sparam="foreground"
    *mv=foregr(foregrndx)
    If *mv\obj[1] ; If there's room for double buffer (2 bmaps)
      rendmain=#FOREG
    EndIf
  EndIf
  If script()\sparam="extraforeground"
    *mv=foregr(foregrndx)
    If *mv\obj[4] ; If there's room for the extra (3 bmaps+2 virtuals)
      rendmain=#XTRAF
    EndIf
  EndIf
  If rendmain
    ; Render in the hidden bitmap...
    db=1-db
    RIAnimInit Start(script()\nparam),*mv\obj[db],rendmain,0,0
    While fdcounter<=framedelay:VWait:Wend ; Wait a while
    If rendmain=#XTRAF ; Now display the rendered bitmap
      *mv\idndx=db+3
    Else
      *mv\idndx=db      ;
    EndIf
    fdcounter=0      ; start the counter for the next frame and...
    CopyBitMap *mv\obj[db],*mv\obj[1-db] ; prepare for animation.
    frame=1
    playmain=1
  EndIf
;
Case "FREE_ANIM"
  FreeBank script()\nparam
;
Case "STOP_MAIN_ANIM"
  playmain=0
;
Case "CONTINUE_MAIN_ANIM"
  playmain=1
;
Case "SET_MAIN_FPS"
  framedelay=60/script()\nparam
;
Case "SPRITES_IN_FRONT"
  ; In front: $0024
  If spritespri=1 DisplayControls 0,$0024,0,0
  If spritespri=2 DisplayControls 0,$0020,0,0
  spritespri=0
;
Case "SPRITES_IN_BACK"
  ; In Back: $0000
  If spritespri=0 Then DisplayControls 0,$0024,0,0
  If spritespri=2 Then DisplayControls 0,$0004,0,0
  spritespri=1
;
Case "SPRITES_IN_MID"
  ; In  Mid: $0004
  If spritespri=0 Then DisplayControls 0,$0020,0,0
  If spritespri=1 Then DisplayControls 0,$0004,0,0
  ;CopperTrace Addr CopList(0),28:Stop
  spritespri=2
;
```

```
Case "MAIN_ANIM7"
  If srcpalette=#DPFSRCPAL Then animop7flag=1 ; Only standard DPF Mode
;
Case "DISPLAY_ON"
  DisplayPalette 0,0 ; To avoid some colour flicker
  BLITZ:QAMIGA
;
Case "DISPLAY_OFF"
  irqflag=0
  AMIGA
;
Case "DISPLAY_SYSTEM"
  irqflag=0
  AMIGA                          ; Restore system display
  VWait script()\nparam          ; wait a while...
  BLITZ:QAMIGA                   ; and return...
  irqflag=1                      ; enabling all the stuff.
;
Case "LOAD_MUS"
  actmus=LoadPTModule (script()\nparam,script()\sparam)
;
Case "PLAY_MUS"
  If actmus Then PlayPTModule script()\nparam
;
Case "MUS_VOL_FADE" ; 1st.P=New volume, 2nd.P=Fade value
  musfadeval=Val(script()\sparam)/64 ; values from 1 to 64
  If musfadeval>1 Then musfadeval=1 ; check for max. value
  If script()\nparam<musvol Then musfadeval*(-1) ; +/- accordly to actual vol.
  newmusvol=script()\nparam ; Start the volume fade!
;
Case "MUS_VOL" ; Set new vol. without fades (override the IRQ mus. fade)
  musfadeval=0:newmusvol=script()\nparam:musvol=newmusvol
  SetPTVolume musvol
;
Case "FREE_MUS"
  If actmus
    StopPTModule
    Free PTModule script()\nparam
  EndIf
;
Case "SOUND_ON" ; Permit sound samples in one channel
  musfadeval=0:musvol=newmusvol ; Stop volume fades!
  SetPTVolume musvol            ; Force new vol.
  SetPTMask %1110
;
Case "SOUND_OFF" ; No sound samples
  SetPTMask %1111
;
Case "EXECUTE"
  Execute_ script()\sparam,0,0
;
Case "SYSTEM_POINTER"
  If script()\sparam="default"
    CopyFile "ENV:Sys/Pointer.bak","ENV:Sys/Pointer.prefs"
  Else
    CopyFile script()\sparam,"ENV:Sys/Pointer.prefs"
  EndIf
  ;NPrint "x=",value MOD 640
  ;NPrint "y=",Int(value/640)
  AbsMouse script()\nparam MOD 640,Int(script()\nparam/640)
;
Case "AUTO_FG_PAL_FADE"
  If srcpalette=#DPFSRCPAL  ; For now only if XFADES are disabled.
    autofgpalfade=1
  EndIf
;
Default
  irqflag=0
```

```
    AMIGA
    EZRequest "AT-Command ERROR!","Script part: "+Str$(Peek.l(atsignaldir))+"|Command:
"+script()\name,"OK"
    BLITZ:QAMIGA
    irqflag=1
  End Select
  .AutomatedEv
  ; @ ============== BEGIN OF AUTOMATED EVENTS ===============
  Repeat
    ; @ ********** automated image load **********
    *mv=backgr(backgrndx)
    If *mv\fready ; Frame ready signal!
      *mv\fready=0
      ;;;;bgnameini=1
      bgnamefin=Instr (bgframe$,",",bgnameini) ; Find the next name...
      If bgnamefin                ; the next name exists? take it,
        name$=Mid$(bgframe$,bgnameini,bgnamefin-bgnameini)
        bgnameini=bgnamefin+1
      Else
        name$="NULL.iff" ; or use an 'empty' image.
      EndIf
      If *mv\y=0                  ; If the mov. is RIGHT/LEFT
        If BLoad (name$,0) ; Bank 0 is for images only!!
          UnpackIFF Start(0),*mv\obj[0],#SCRHEIGHT,(bgframec-1)*(#SCRWIDTH/8)
        EndIf
        If bgframec=4 ;
          Use BitMap *mv\obj[0]
          UnpackIFF Start(0),*mv\obj[0],#SCRHEIGHT,(0)*(#SCRWIDTH/8)
          ;BlockScroll #SCRWIDTH*3,0,#SCRWIDTH,#SCRHEIGHT,0,0
          ; repeat the image for the next 1234 cycle: 1=4
        EndIf
      Else                        ; If the mov. is UP/DOWN
        If BLoad (name$,0) ; Bank 0 is for images only!!
          UnpackIFF Start(0),*mv\obj[0],#SCRHEIGHT,(bgframec-1)*(#SCRWIDTH/8)*#SCRHEIGHT
        EndIf
        If bgframec=4 ;
          Use BitMap *mv\obj[0]
          UnpackIFF Start(0),*mv\obj[0],#SCRHEIGHT,(0)*(#SCRWIDTH/8)*#SCRHEIGHT
          ;BlockScroll 0,#SCRHEIGHT*3,#SCRWIDTH,#SCRHEIGHT,0,0
          ; repeat the image for the next 1234 cycle: 1=4
        EndIf
      EndIf
      ILBMPalette Start(0),#BACKG ; New palette 1 for background
      bgframec+1                  ; Load frames to right/down
      If bgframec>4 Then bgframec=2 ; 234 234 234 ...
    EndIf
    ;
    *mv=foregr(foregrndx)
    If *mv\fready ; Frame ready signal!
      *mv\fready=0
      If autofgpalfade=1 ; Fade colours from one frame to the next...
        For ini=1 To 15:CopyColour #FOREG,#DPFSRCPAL,ini,ini+16:Next ini ; GetFG
        XFadeInit(XFWork,0,#DPFSRCPAL,17,15,*mv\fmovs)
        xfadenum=*mv\fmovs
      EndIf
      ;;;;fgnameini=1
      fgnamefin=Instr (fgframe$,",",fgnameini) ; Find the next name...
      If fgnamefin                ; the next name exists? take it,
        name$=Mid$(fgframe$,fgnameini,fgnamefin-fgnameini)
        fgnameini=fgnamefin+1
      Else
        name$="NULL.iff"          ; or use an 'empty' image.
      EndIf
      If *mv\y=0                  ; If the mov. is RIGHT/LEFT
        If BLoad (name$,0) ; Bank 0 is for images only!!
          UnpackIFF Start(0),*mv\obj[0],#SCRHEIGHT,(fgframec-1)*(#SCRWIDTH/8)
        EndIf
        If fgframec=4 ;
```

```
      UnpackIFF Start(0),*mv\obj[0],#SCRHEIGHT,(0)*(#SCRWIDTH/8)
      ;BlockScroll #SCRWIDTH*3,0,#SCRWIDTH,#SCRHEIGHT,0,0
      ; repeat the image for the next 1234 cycle: 1=4
    EndIf
  Else                        ; If the mov. is UP/DOWN
    If BLoad (name$,0) ; Bank 0 is for images only!!
      UnpackIFF Start(0),*mv\obj[0],#SCRHEIGHT,(fgframec-1)*(#SCRWIDTH/8)*#SCRHEIGHT
    EndIf
    If fgframec=4 ;
      UnpackIFF Start(0),*mv\obj[0],#SCRHEIGHT,(0)*(#SCRWIDTH/8)*#SCRHEIGHT
      ;BlockScroll 0,#SCRHEIGHT*3,#SCRWIDTH,#SCRHEIGHT,0,0
      ; repeat the image for the next 1234 cycle: 1=4
    EndIf
  EndIf
  ILBMPalette Start(0),#FOREG ; New palette 2 for foreground
  fgframec+1                  ; Load frames to right/down
  If fgframec>4 Then fgframec=2 ; 234 234 234 ...
EndIf
;
fin=0
For act=virtgr(virtgrndx)\first To virtgr(virtgrndx)\last
  *mv=sprt(act)
  If *mv\fready ; Frame ready signal!
    fin=1
    *mv\fready=0
    ;;;;vgnameini=1
    vgnamefin=Instr (vgframe$,",",vgnameini) ; Find the next name...
    If vgnamefin                ; the next name exists? take it,
      name$=Mid$(vgframe$,vgnameini,vgnamefin-vgnameini)
      vgnameini=vgnamefin+1
    Else
      name$="NULL.sp"           ; or use an 'empty' SPRITE IMAGE.
    EndIf
    LoadSprites *mv\obj[1-*mv\idndx],*mv\obj[1-*mv\idndx],name$
    LoadSprites *mv\obj[1-*mv\idndx]+1,*mv\obj[1-*mv\idndx]+1,name$+"o"
    ; In the hidden sprite! (1-db)
    ; IMPORTANT-The interlaced sprites must be saved:even(0)+odd(1)
  EndIf
Next
If fin
  vgnamefin=Instr (vgframe$,",",vgnameini) ; Now load palette if needed
  If vgnamefin                             ; This name exists ?
    name$=Mid$(vgframe$,vgnameini,vgnamefin-vgnameini)
    vgnameini=vgnamefin+1
    If name$<>"NULL.pal" Then LoadPalette #VIRTG,name$
  EndIf
EndIf
;
; @ ** ANIMATION HANDLER (next frame accord to FPS,etc.) **
If playmain<>0 AND rendmain<>0 AND fdcounter>framedelay
  fdcounter=0
  If rendmain=#BACKG Then *mv=backgr(backgrndx)
  If rendmain=#FOREG OR rendmain=#XTRAF Then *mv=foregr(foregrndx)
  Use BitMap *mv\obj[db]
  db=1-db                         ; Render in hidden bitmap
  RINextAnimFrame *mv\obj[db]     ;
  frame+1
  If rendmain=#XTRAF ; show this recently rendered frame!
    *mv\idndx=db+3
  Else
    *mv\idndx=db      ; show this recently rendered frame!
    If animop7flag    ; and Anim7 props. if needed (standard DPF mode for now)
      Filter On
      For ini=0 To 15:CopyColour #FOREG,#DPFSRCPAL,ini,ini+16:Next ini
      DisplayPalette 0,#DPFSRCPAL
      ;DuplicatePalette #DPFSRCPAL,0
      ;DisplayPalette 0,0
    EndIf
```

```
      EndIf
      VWait ; take a while before next loop ;
    Else                                  ; This also free more CPU...
      VWait ; take a while before next loop ;
    EndIf
    ;
    ; @ *** Wait events (delay to get next instruction)
    If RawStatus($66) ; FFWD key (LAMIGA)?
      waittime=0:waitflag=0:xfadenum=0:fade=-1  ; Next script command.
    EndIf
    If waittime
      If counter>waittime Then waitflag=0:waittime=0 ; go to next script command.
    EndIf                                          ; WAIT TIME IS OVER...
    If waitframe                                   ;
      If frame=waitframe Then waitflag=0:waitframe=0 ; go to next script command.
    EndIf                                          ;
    If waitmov                                     ;
      If movtimer>waitmov Then waitflag=0:waitmov=0  ; go to next script command.
    EndIf
  Until waitflag=0
  ; ================== END OF AUTOMATED EVENTS =================
  act=NextItem(script()) ; Exit if it's the END command or the very last command.
  If script()\name="END" Then act=0
Wend
;
; @ Send the END_OF_PART signal to the ATdaemon.
Poke.l(atsignaldir),$FFE
Forever ; Repeat main loop forever, command equivalent: until 0
;

.___SUBRs___
; @ * SUB-ROUTINES (For speed, but can be statements/funcions) *
.ColFunction
LoadDPFpalB:
If endreg<>1 Then endreg=16
colregy=0
For y=1 To 16
  colregx=colregy
  For x=1 To endreg
    ;Print colregx:Print "-"
      CopyColour 1,4,y-1,colregx ;Copy from Pal1(Bg) to Pal4(255)
    Gosub INCcolregx
  Next x
  Gosub INCcolregy
  ;NPrint ""
Next y
endreg=16
Return

LoadDPFpalF:
If endreg<>1 Then endreg=16
colregx=0
For x=1 To 16
  colregy=colregx
  For y=1 To endreg
    ;Print colregy:Print "-"
    If customdepth=1
      If x>8 Then CopyColour 3,4,1,colregy ;From Pal3 Reg1 to Pal4 regs.8-15
    Else
      CopyColour 2,4,x-1,colregy ;Copy from Pal2(Fg) to Pal0(255)
    EndIf
    Gosub INCcolregy
  Next y
  Gosub INCcolregx
  ;NPrint ""
Next x
endreg=16
customdepth=0
```

```
Return

INCcolregy:
inc=1
countx+1
If y=1 Then countx=1
If y=8
  inc=43
  countx=0
EndIf
If countx=2 Then inc=3
If countx=4
  inc=11
  countx=0
EndIf
colregy=colregy+inc
Return

INCcolregx:
inc=2
county+1
If x=1 Then county=1
If x=8
  inc=86
  county=0
EndIf
If county=2 Then inc=6
If county=4
  inc=22
  county=0
EndIf
colregx=colregx+inc
Return

.____IRQ____
;****************** BEGIN OF IRQ CODE ******************
.VBlank_IRQ:
; @ Real time critic actions every (1/60)s
; Related variables:
; @ irqflag   -   ON/OFF
; backgr(),foregr(),virtgr(),sprt() - GFX movement management
; backgrndx - Currently used movement (index)
; foregrndx -      "
; virtgrndx -      "          group of movs.
; @ counter, fdcounter, movtimer      - Custom timer update
; paintfill, paintfillsp              - Line scan FX management
; fade, xfadenum, fadespeed           - Colour FX management
; musvol, newmusvol, musfadeval       - Music tolume management

; @ The next are PRIVATE! for the IRQ 5 routine.
DEFTYPE .movement *irq,*ir2 ; Speed up access to mov. structs.
DEFTYPE .l sc               ; Sprite Counter

AMIGA ; The IRQ must be enabled in system mode.

; @ VBlank server
SetInt 5     ; IRQ number 5 (VBlank server)
If irqflag=1 ; If ACTIVE irqflag!!!!!!!!!!!!!!!!!!!
;MOVE.w $ffff,$dff180
;
counter+1    ; general purpose timer
fdcounter+1  ; frame delay timer active
movtimer+1   ; time of the actual movement
;
; @ Copper list update
; *** PAINT-FILL effect ***
If paintfill>0 AND paintfill<201 ; (could be negative but it's unfinished...)
  CopperReset 0,0,6
```

```
  CopperWait 0,paintfill+44
  ;paintfill+paintfillsp          ; See down (*)
EndIf

; ® Movement structures update
; *** GRAPHIC movements ***
*irq=backgr(backgrndx):Gosub IRQmovement
*irq=foregr(foregrndx):Gosub IRQmovement
For sc=virtgr(virtgrndx)\first To virtgr(virtgrndx)\last
*irq=sprt(sc):Gosub IRQmovement
Next sc

; #---INTERLACED routine: (sprite ID's must be PAIR's)
  If Peek($dff004)<0  ; ----- 1st laced line -----

*irq=backgr(backgrndx)
*ir2=foregr(foregrndx)
DisplayBitMap 0,*irq\obj[*irq\idndx],*irq\x,*irq\y-0,*ir2\obj[*ir2\idndx],*ir2\x,*ir2\y
For sc=virtgr(virtgrndx)\first To virtgr(virtgrndx)\last
*irq=sprt(sc)
DisplaySprite 0,*irq\obj[*irq\idndx]+*irq\yodd,*irq\x,*irq\ymid + *irq\yodd,*irq\sprtch;1stL
Next sc
  Else                ; ----- 2nd laced line -----

If paintfill>0 AND paintfill<201 ; (*) In 2nd interlaced line to avoid aberrations.
  paintfill+paintfillsp
EndIf

*irq=backgr(backgrndx)
*ir2=foregr(foregrndx)
DisplayBitMap 0,*irq\obj[*irq\idndx],*irq\x,*irq\y+1-0,*ir2\obj[*ir2\idndx],*ir2\x,*ir2\y+1
For sc=virtgr(virtgrndx)\first To virtgr(virtgrndx)\last
*irq=sprt(sc)
DisplaySprite 0,*irq\obj[*irq\idndx]+(1-*irq\yodd),*irq\x,*irq\ymid,*irq\sprtch        ;2ndL
Next sc
  EndIf               ; ----- ------------- -----

; #---NON-LACED routine:
; #Empty.

;*** Fade In/Out ***
If fade>=0 AND fade<=1 ; fade=1 : Fade OUT, fade=0 : Fade IN
;  MOVE.w $0f0,$dff180
  FadePalette srcpalette,0,fade  ; Slooow default routine!
;  MOVE.w $000,$dff180
  DisplayPalette 0,0
  fade=fade+fadespeed  ; (fadespeed can be negative)
Else
  If xfadenum>0        ; Until requested number of xfades is finished.
;    MOVE.w $0f0,$dff180
    XFade{XFWork,0}    ; Fast ML XFade!
    xfadenum-1         ; one requested less
;    MOVE.w $000,$dff180
    DisplayPalette 0,0
  EndIf
EndIf

;*** Music volume fade (0-64) ***
If Int(musvol)<>newmusvol ; If the new volume hasn't been reached ...
  musvol+musfadeval  ; (musfadeval - is float(q) and can be negative if needed)
  SetPTVolume musvol ; musvol is float (quick) for fine increments.
EndIf
;
;MOVE.w $0000,$dff180
EndIf ; END irqflag!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
End SetInt
Return
```

```
.IRQmovement:
; @ *irq, *ir2 used for movements in backgr,foregr,virtgr
If *irq\loop>0
  If *irq\nmovs>0 ; number of movs to do in every loop > 0 ?
    If *irq\idmovs>0 Then *irq\idmov+1 ; inc. id index if id movs > 0...
    If *irq\idmov>*irq\idmovs          ; and make the double/multi buffer!
      *irq\idmov=1:*irq\idndx+1:If *irq\idndx=*irq\nids Then *irq\idndx=0
    EndIf ;       ^ Due QWrap and scroll by blocks!
    *irq\nmov+1                        ;
    If *irq\nmov>*irq\nmovs            ; go to next loop...
      *irq\nmov=1:*irq\loop-1          ; and put initial coords. if more loops...
      If *irq\loop<>0 Then *irq\x=*irq\xini:*irq\y=*irq\yini
    EndIf ;       ^  Due QWrap and scroll by blocks!
    *irq\x=*irq\x+*irq\xvel ; Sum. the speed/acc.
    *irq\y=*irq\y+*irq\yvel
    *irq\xvel+*irq\xa       ; ...
    *irq\yvel+*irq\ya
    *irq\x=QWrap(*irq\x,*irq\left,*irq\right) ; adjust the coords.
    *irq\y=QWrap(*irq\y,*irq\up,*irq\down)    ; adjust the coords.
    If sprtch>-1 ; SPRITE purposes
      *irq\ymid=Int(*irq\y)/2 ; Sprite resolution in LACED mode is (/2).
      If QFrac(*irq\ymid)<>0 Then *irq\yodd=1:Else *irq\yodd=0 ; impar?adjust sprites
    EndIf
    If *irq\fmovs>0 ; If it need load frames
      *irq\fmov+1
      If *irq\fmov>*irq\fmovs
        *irq\fmov=1:*irq\fready=1
      EndIf ;       ^ Due Qwrap and scroll by blocks!
    EndIf
  EndIf
End If
Return
;***************** END OF IRQ CODE **********************
;
.___E_N_D___
```

.Structures

Fue creada una estructura para implementar movimientos en dos dimensiones para los dos despliegues y el despliegue virtual de *sprites*. Estos movimientos requieren algunos parámetros iniciales que deben indicar:

Número de *bitmaps* para el despliegue (2 si se requiere un buffer doble para evitar parpadeos, etc.): nids

Límites de movimiento: right, down, left, up

Coordenadas de inicio: xini, yini

Velocidad y aceleración (en este caso, se utilizan dos factores de aceleración para realizar algunos movimientos de tipo parabólico cuando se requiera): xv, yv, xa, ya

Número de movimientos: idmovs, fmovs, nmovs, loops

El número de estructuras posibles debe relacionarse con el espacio disponible en memoria, pero por ahora es un arreglo de *100 elementos para cada movimiento: backgr, foregr, sprt (200). Esto significa* que podemos tener 100 distintos efectos de video para cada entidad gráfica (200 para los *sprites*).

.StartupPars

Como ya se mencionó, se requiere una forma de comunicación al exterior, por ahora solo utilizamos el camino más sencillo de obtener una dirección de memoria (long) como parámetro. Esta dirección de memoria debe ser previamente asignada por un programa externo para que sirva como un canal o señal de comunicación entre ambos.

._P_R_O_C_S_ y ___SUBRs___

Estos procedimientos y subrutinas se utilizan para realizar algunas operaciones críticas con los colores de las imágenes, no es un proceso de imagen real en esencia, más bien son operaciones y trucos sobre el hardware, que ofrecen resultados similares, solo que en tiempo real. Algunos de estos trucos no han sido implementados previamente en ningún otro programa para esta plataforma, para obtener el máximo rendimiento se utiliza el lenguaje ensamblador.

.__M_A_I_N__

Primeramente se inicializan algunas secciones de memoria relacionados con los objetos gráficos, posteriormente se definen las constantes que contienen los valores más usados en todo el programa y finalmente se inicializan algunas variables relacionadas con los efectos extra de video implementados hasta ahora: xfadenum, fade, paintfill.

.MakeDisplay

Anteriormente se mencionó que esta arquitectura utiliza un coprocesador -Copper- sobre el que recae la responsabilidad para generar el despliegue gráfico en pantalla, en este caso primero se utilizan algunas instrucciones implementadas en el lenguaje que realizan un programa "esqueleto" para el coprocesador.

Después de indicar algunas modificaciones personalizadas a los registros del hardware, agregamos un pequeño programa de instrucciones propias del coprocesador Copper que nos ayudarán a realizar algunos efectos gráficos de video en sincronía con el barrido electrónico de la pantalla.

.InstallIRQ, .____IRQ____ y .IRQmovement

Esta es la parte del programa que se encarga de realizar las acciones automatizadas críticas en tiempo real, para esto utilizamos una facilidad del hardware que activa una interrupción durante el intervalo de borrado horizontal (*vertical blanking*, en NTSC=60 Hz), antes de que se genere el nuevo cuadro de video.

En este sentido, esta sección del código, que puede ser activada o desactivada a voluntad mediante la variable: irqflag, se ocupará de procesos críticos como la actualización de contadores de tiempo (counter, movtimer, fdcounter), actualización del programa para el Copper que permite los efectos de video en sincronía con el barrido electrónico y finalmente actualización de las estructuras de control de movimientos para cada despliegue, mediante dos apuntadores a estas estructuras: *irq e *ir2, que permiten un acceso más rápido que el uso de índices. Estos apuntadores permiten acceder y modificar los elementos clave de cada estructura de movimiento (mediante la subrutina .IRQmovement) para generar el movimiento gráfico de las entidades de despliegue básicas hasta ahora definimos: backgr, foregr y virtualgr.

.MovObjs

En el futuro cada estructura de movimiento podrá ser leída desde disco, por ahora, estas son incluidas y llenadas dentro del programa, en este caso se utiliza una para cada parte del video final.

.Script

Por el contrario, el script sí es leído desde disco para hacer el proceso más versátil, en busca de velocidad, cada comando con sus parámetros es leído desde el archivo de texto, filtrado y posteriormente almacenado en memoria dentro de una lista, por ahora limitada a 2000 elementos como máximo, con un acceso más veloz que cualquier arreglo indizado y ocupando de manera óptima solo la memoria necesaria, incluso para las cadenas.

.___BEGIN___

Aquí comienza el ciclo principal del programa propiamente dicho.

Aunque el número de objetos gráficos depende de la memoria libre, es necesario llevar algún orden para no utilizar el mismo identificador en dos objetos distintos, en este caso se definieron dos listas de enteros largos (freebitmaps y free sprites), que utilizaremos a manera de pilas. Primeramente se almacenan todos los identificadores disponibles, de este modo, durante la ejecución del programa se remueve un identificador de la pila cada vez que se requiere, pero al dejar de utilizarlo se debe regresar a la misma.

En las definiciones de las variables relacionadas, cabe hacer notar que existen variables repetidas con el prefijo: new_, esto es debido a que podemos construir un nuevo (new_) despliegue mientras aún se está visualizando el actual, de este modo, se eliminan las posibles pausas.

Posteriormente se espera la señal que indicara la parte del script a ejecutar, es necesario utilizar un VWait (Espera al inicio del intervalo de borrado vertical) para no consumir todos los recursos del CPU mientras se espera, de cualquier modo, esto permitirá un chequeo 60 veces por segundo, lo cual es más que suficiente.

A continuación se busca la primera instrucción de la parte del script seleccionada (instrucción: PART #). Si ésta no existe se envía una señal de salida del programa ($FFF) y termina el proceso.

Si existe se inicia un ciclo que lee la siguiente instrucción y compara con un select/case básico para ejecutar el comando adecuado en tiempo real.

En la parte final del ciclo, en donde se realizan los eventos o acciones automatizadas, se cargan nuevas imágenes si el movimiento del despliegue en cuestión lo requiere, se genera el siguiente cuadro de animación si el número de cuadros por segundo está alcanzado, etc.

Finalmente se verifican los comandos de espera para saber cuando volver al ciclo principal de lectura de la siguiente instrucción.

Cuando se termina con la parte en cuestión del script, se envía una señal de finalización de la parte en cuestión, es decir, se borra o se limpia la señal anterior con un valor ($FFE).

En el Apéndice 4 se muestra una aproximación del script final utilizado para el sistema multimedia/video de este trabajo, los comandos no se explican porque es más fácil entenderlos haciendo pruebas de su uso que de cualquier otra forma:

6.4.2.3 Programa 'daemon' para establecer la comunicación

6.4.2.3.1 Algoritmo de solución

En el punto 6.4.2.1 se explicó, planteó, y solucionó parcialmente el problema de la comunicación con el programa ScalaMM, a grandes rasgos, se requiere de un programa que comunique a los dos programas principales: nuestro programa, también llamado AT, y en este caso: atplayer y el programa ScalaMM, llamado en este caso: mmplayer. A continuación se implementa la solución para el programa de comunicación, que ahora tiene como nombre clave: atdaemon

Fig.26

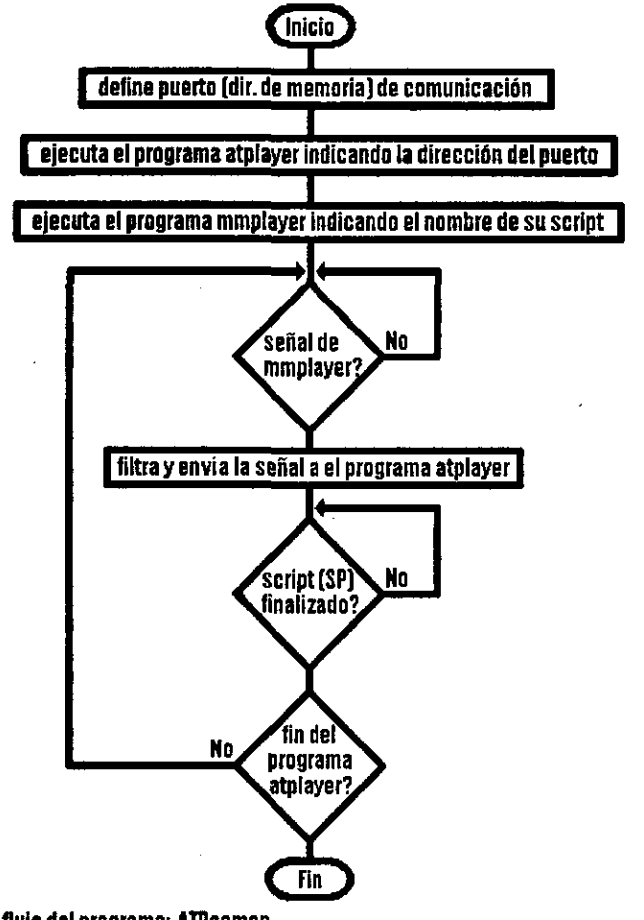


Diagrama de flujo del programa: ATDaemon

6.4.2.3.2 Listado final y explicaciones

Nota: Los listados finales están comentados en inglés para poder distribuirlos en Internet si se requiere.

```
#include <exec/exec.h>
#include <exec/types.h>
#include <dos/dos.h>
#include <graphics/gfxbase.h>
#include <graphics/copper.h>
#include <graphics/view.h>

#include <stdio.h>
#include <string.h>

#include <proto/exec.h>
#include <proto/dos.h>
#include <proto/graphics.h>

char *getarg (char *s)
{
    printf("A.T. daemon V0.1 (C)1998 by Carlos Villarroel\n");
    printf("Usage: atdaemon <MM-SCRIPT>\n");
    printf("Type in path & name of MM-SCRIPT: ");
    scanf("%s",s);
    return s;
}

struct GfxBase *GfxBase;

// @ Parameter: path/name of mmplayer script
main(int argc, char *argv[])
{

/* UWORD = unsigned word (unsigned short: 16 bits) */
UWORD cmparray[] = {
    0x8f01,0xfffe,0x0106,0x0ca2,
    0x0180,0x0000,0x0182
    }; /*0x8e01|0x8f01*/

    char atcall[50] = "run atplayer ";
    char atarg[12];
    char mmcall[100] = "run Work:Multimedia/Scalamm300/scalammplayer ";
    char mmarg[40];

    // @ 32 bit values by default for pointers & integers (short to 16bit)
    int atsignal;
    int signal=0,oldsignal=0,sigfound=FALSE; /* sigfound=0 */

    // @ 32 bit long to store atsignal mem. pos.
    unsigned long atport;

    register unsigned short *cmpndx,*cprlst;

    if ( argc < 2 ) strcat(mmcall,getarg(mmarg));
```

```
else strcat(mmcall,argv[1]);

printf("creating.... atport\n");
atport = (unsigned long)&atsignal; // @ store and convert
stcul_d(atarg,atport); /* This converts from ULONG to DEC.STRING */
strcat(atcall,atarg); // @ generates: "run atplayer <MEMPOS>"

// @ Execute: atplayer and mmplayer programs
atsignal=0x0FFE; /* Begin AT with a pause signal */
/* strcat(atcall," >NIL:"); */
printf("executing... %s\n",atcall);
if ( Execute (atcall,NULL,NULL) == FALSE ) return -1;
Delay(180);
if ( strcmp( " 0",strrchr(mmcall,' ') ) )
{           /* If the mmplayer parameter<>0 then execute it */
    printf("executing... %s\n",mmcall);
    if ( Execute (mmcall,NULL,NULL) == FALSE ) return -1;
}

if ( GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0) )
{
// @ Main loop checking for END_OF_PROGRAM signal from atplayer
    while (atsignal != 0x0FFF)/* 0x0FFF=END_OF_PROGRAM */
    {
// @ Read mmplayer signal (from copper list) every frame
        /* Try to read an MM signal */
        while (signal == oldsignal)
        {
            sigfound = FALSE ; /* Assume this to found a new signal */
            while (sigfound == FALSE) /* checking active copper list */
            {
                WaitTOF();              /* every frame */
                /* Check & clear BREAK-CTRL_C signal (0LONG,MASK)*/
                if(SetSignal(0L,SIGBREAKF_CTRL_C) & SIGBREAKF_CTRL_C)
                {
                    printf("BREAKed!\n");
// @ End if CTRL-C S.O. signal
                    goto endofprog;
                }
// @ Find the copper list of current/active display
                cprlst = GfxBase->ActiView->LOFCprList->start;
// @ Compare initial copper instructions with our array: cmparray
                sigfound=TRUE; /* Assume it's true until we found... */
                cmpndx=cmparray; /* As this is a purely ascending access
                                    to the -cmparray- we can speed it up
                                    using the register based pointer:
                                    -cmpndx-. Also we are saving one
                                    variable... */
                while (*cmpndx != 0x0182) /* ...pos of signal - 1  */
                {
                    /* printf("arr=%d cpr=%d\n",*cmpndx,*cprlst); */
                    if (*cmpndx != *cprlst) /* Or again if they differ */
                    {  sigfound = FALSE; /*printf("bad\n");*/ break;  }
                    cmpndx++;  /* point to next element... */
```

```
                    cprlst++;
                }
            }
            cprlst++;
            signal = *cprlst;
        } /* Until there is a new MM signal */
        oldsignal = signal;
    // @ Signal found, now send it to atplayer
        /* The send the atsignal */
        atsignal = signal;
        printf("atsignal=%d\n",atsignal);
    // @ And wait (every frame) for one of the atplayer endsignals
        while (atsignal < 0x0FFE) WaitTOF(); /* 0x0FFE=END_OF_PART */
    }
    // @ CTRL-C or atplayer END_OF_PROGRAM signal:
    endofprog:
    atsignal=0x0FFF; /* Force atplayer to exit if atdaemon is breaked */
    Delay(60); /* Wait a second to ensure atplayer gets the signal */
    CloseLibrary( (struct Library *)GfxBase );
}
return 0;
}
```

El programa: atdaemon está escrito en 'C' para aprovechar la generación de código óptimo en el modelo pequeño del compilador, además de otras ventajas como las variables registro, lo cual nos ofrece alguna ganancia en velocidad a comparación de cualquier otro lenguaje.

Primeramente el programa recibe como parámetro la ruta y el nombre de un script del programa ScalaMM, este script será el encargado de controlar al programa: atplayer.

Posteriormente se define la posición de memoria que servirá como puerto de comunicación entre el programa: atdaemon y el programa atplayer (nuestro programa principal), para esto, simplemente se declara un entero de 32 bits y se encuentra y almacena su dirección de memoria para posteriormente enviarla al programa: atplayer

Nota: En esta plataforma el sistema operativo y el hardware siempre han manejado 32 bits, y en casi todos los compiladores para esta plataforma, los tipos de datos se basan en estos 32 bits, así que existen ligeras diferencias con respecto a otras arquitecturas que se muestran en la siguiente tabla:

Tipo	Bytes	Mínimo	Máximo
signed char	1	-128	+127
unsigned char	1	0	255
signed short	2	-32,768	+32,767
unsigned short	2	0	65,535
signed int	4	-2,147,483,648	+2,147,483,647
unsigned int	4	0	4,294,967,295
signed long	4	-2,147,483,648	+2,147,483,647
unsigned long	4	0	4,294,967,295
float (IEEE)	4	3.402E-37	3.402E+38
double (IEEE)	8	2.222E-308	1.797E+308
float (FFP)	4	5.421E-20	9.223E+18
double (FFP)	4	5.421E-20	9.223E+18

También los apuntadores miden 4 bytes de longitud y los valores por defecto de los enteros son de 32 bits, por lo que en este caso simplemente declaramos un entero. De cualquier modo el compilador tiene una opción (shortint) que permite manejar los valores de dos bytes por defecto, lo cual incrementa la posibilidad de transferir el código si así se requiere, en este caso es altamente improbable.

Después de que se ejecutan los programas correspondientes, mediante la función: Execute() del sistema operativo, comienza el ciclo principal que verifica la señal de salida (0x0FFF) que pudo haber enviado el programa: atplayer en el momento en que éste finaliza o sale del sistema.

A continuación se asigna la dirección de memoria en donde se encuentra la lista o programa COPPER del despliegue actual o activo a un apuntador hacia una palabra de 16 bits: cprlst = GfxBase->ActiView->LOFCprList->start; (Para separar fácilmente las instrucciones del Copper, cada una consta de 16 bits), éste apuntador y el apuntador: cmpndx, que apunta a un arreglo que contiene los valores clave para identificar una señal del programa mmplayer, son incrementados para realizar la identificación mediante una comparación básica de sus valores (*cmpndx != *cprlst). Todo esto se realizó mediante apuntadores para ganar algo de velocidad con respecto a la comparación de un arreglo indizado.

Posteriormente se envía la señal encontrada hacia el programa: atplayer (atsignal = signal) y finalmente se espera la señal de respuesta (0x0FFE) del mismo programa: atplayer que indicará el momento en que termina de ejecutar la parte del script correspondiente.

Es importante mencionar que durante cada ciclo de espera se utiliza la función: WaitTOF() del sistema operativo, que es equivalente al comando: VWait utilizado en BBasic, para liberar el CPU, de cualquier modo, también aquí esto permitirá un chequeo 60 veces por segundo, lo cual es más que suficiente.

Para poder abortar los programas: atdaemon y atplayer en cualquier momento, se utilizan las funciones de comunicación y señales incluidas en el sistema operativo: SetSignal(0L,SIGBREAKF_CTRL_C) /* 0Long,Mask */ regresa un valor verdadero si la combinación de teclas: CTRL+C está activa para el proceso correspondiente (por ejemplo, en su ventana), de este modo, podemos enviar la señal de salida (0x0FFF) hacia el programa: atplayer y forzar una salida o finalización de ambos procesos. Aquí utilizamos un goto para simplificar las cosas.

6.5 Sincronización final audio/video

El problema de la sincronía y temporización entre video, musicalización y voz se reduce a aumentar o disminuir las pausas en los script's respectivos de los programas: atplayer y mmplayer (nuestro programa y el programa ScalaMM). El hardware de esta arquitectura cuenta con 4 canales de sonido, así que podemos utilizar 1 o dos canales para el audio y el resto para la música. Nuestro programa: atplayer cuenta con una instrucción en el script que permite ejecutar otro proceso que a su vez reproducirá música o audio paralelamente a la presentación gráfica.

Desafortunadamente, el audio requiere mucho espacio en disco duro (más de 500 megas), pero por ahora podemos almacenarlo en un dispositivo externo para mezclarlo con música, esta sí reproducida y controlada por la computadora misma.

Análisis de resultados y conclusiones

Si bien los programas y herramientas de los equipos de cómputo modernos pueden realizar todo esto de una forma bastante sencilla, es también evidente que están fuera de alcance para muchos de nosotros debido al factor económico.

El resultado obtenido finalmente, fue satisfactorio, se alcanzó el objetivo y además se demostró que no es necesario utilizar lo último en tecnología para obtener un resultado aceptable. Basta con saber utilizar a fondo un sistema bien diseñado.

Por el camino, se observó también que las técnicas de programación utilizadas en esta arquitectura son perfectamente aplicables a cualquier otro sistema multiproceso/multiusuario moderno existente, en especial aquellos escritos en lenguaje C.

Aunque hoy en día los niveles de abstracción en las técnicas de programación, por ejemplo la programación orientada a objetos, simplifican mucho el trabajo, nada sustituye el conocimiento básico del funcionamiento interno de la arquitectura y el CPU, finalmente, ningún programa puede generar un código en bajo nivel tan eficiente como el que podría escribir un programador que conoce a fondo el problema. Simplemente hay que considerar cada técnica o lenguaje de programación como una herramienta más, y utilizar la más conveniente para cada parte del problema y de acuerdo al problema en cuestión, poniendo en la balanza todos los factores que influyen en la solución del mismo.

Finalmente el objetivo principal se cumplió, pero hay algo más que podemos considerar de suma importancia:

Hace algún tiempo, cuando apenas las computadoras de 8 bits se introducían a México, leía un libro escrito por uno de los primeros ingenieros en computación mexicanos, el Ing. Ricardo Hernández Jiménez, en este libro, él mencionaba el desarrollo de la informática como la llave para lograr un desarrollo económico y social. Pero para lograr este desarrollo era necesario invertir en la materia prima más abundante en nuestro país: nuestro intelecto. Desafortunadamente, esa inversión se alimenta con educación, y todos sabemos las deficiencias que aún tenemos en este sentido.

Si nuestro objetivo es entrar en la carrera informática para lograr algún desarrollo, la única opción es hacerlo por el camino del software, programas y algoritmos, sin embargo, nos preguntamos: ¿Cuántos mexicanos tienen la oportunidad de programar una computadora? ¿Cuántos mexicanos pueden contar con los equipos más avanzados para tener acceso a los últimos programas y herramientas?

A pesar de la continua reducción de precios en los equipos de cómputo, es extremadamente difícil conseguir un equipo de calidad, que no pueda ser considerado desechable, por menos de N$8000. Pero esto es demasiado para muchos de nosotros, sin considerar el costo de los programas.

Recientemente se mostraban los precios de equipos de cómputo, en dólares, en una dirección de Internet, con lo siguiente: PC PentiumII Multimedia de marca comercial - $1000, Amiga1200 030 - $250 expandible a PPC (sumar $250 por la tarjeta de expansión). Como se observa, la diferencia de precios es muy grande, puesto que ambos equipos pueden realizar las funciones más comunes de forma similar: acceso a Internet, proceso de textos, proceso gráfico, audio, etc. ¿Cómo es posible esto, si el procesador Pentium es muchas veces más rápido que un 030? La respuesta es el sistema operativo, pues el monopolio de las computadoras compatibles con IBM ofrece productos que no son lo más óptimo en cuanto a software. ¿Por qué un procesador de textos, similar y con todas las funciones de Word, ocupa menos de 4 megabytes de memoria en disco duro en una computadora Amiga? Podemos nombrar muchos ejemplos más, y además imaginar el rendimiento de esta arquitectura alternativa con un CPU más poderoso como el PPC, equivalente o más veloz que un Pentium.

Pero sobre todo, lo más importante es el hecho de que existen alternativas computacionales a considerar, perfectamente utilizables para todas las aplicaciones modernas, incluidos los entornos de red, pero que pueden realizar el mismo trabajo a un costo mucho menor que el de cualquier PC o estación de trabajo comercial. En este sentido, regresando al tema de la educación, parece ser que ahora no hay pretexto para que cada salón de clase cuente al menos con una computadora como auxiliar.

Referencias Bibliográficas

· Internet

Sobre todo páginas de universidades con información acerca de video.

Http://www.funet.fi/pub/hardware (Finlandia)
Http://www.funet.fi/pub/amiga (Finlandia)
Http://www.hut.fi/Misc/Electronics/video.html (Finlandia)
Http://ftp.unina.it/pub/electronics (Italia)
Http://wuarchive.wustl.edu/~aminet/docs/hard/video (Norteamérica)
Http://www.blackdown.org/~hwb/hwb.html (Página con información sobre hardware)

Y algunas páginas más sobre la plataforma: AMIGA que pueden ser localizadas a partir de la dirección siguiente:

Http://www.cucug.org (Directorio de páginas relacionadas con la plataforma: Amiga)

· Bibliografía

SCHAUN, Dirk: Amiga C, Abacus, USA 1988

COMMODORE AMIGA INC.: Amiga Hardware Reference Manual (revised and updated), Addison-Wesley Publishing Inc., USA 1989

COMMODORE AMIGA INC.: Amiga ROM Kernel Reference Manual: Includes and Autodocs (revised and updated), Addison-Wesley Publishing Inc., USA 1989

COMMODORE AMIGA INC.: Amiga ROM Kernel Reference Manual: Libraries and Devices (revised and updated), Addison-Wesley Publishing Inc., USA 1989

Apéndice 1: Lista de funciones principales del sistema operativo

La lista de TODAS las funciones *disponibles y sus parámetros para cada librería* se encuentran en la documentación oficial, de cualquier modo, aquí se listan las funciones más importantes de las librerías más utilizadas: EXEC, DOS, GRAPHICS, INTUITION y DISKFONT.

Estas funciones siempre regresan un número de 32 bits (long) que puede ser verdadero (función ejecutada con éxito) o falso (error), o bien un valor relacionado con la rutina.

Un * que precede a la función significa que ésta es privada y no debe ser llamada desde el lenguaje.

Para llamar a estas funciones desde lenguaje ensamblador, también se incluyen los desplazamientos relativos (offsets) a partir de la dirección base de la librería y los registros del CPU que deben contener los parámetros antes de la llamada.

Nota: Los comentarios de esta compilación están en el idioma inglés, *para conservar las ideas* y significados originales y además poder distribuirlos en Internet si así se requiere.

EXEC

-30 Supervisor(userFunction)(a5)

---- special patchable hooks to internal exec activity ---

-36 *execPrivate1()()
-42 *execPrivate2()()
-48 *execPrivate3()()
-54 *execPrivate4()()
-60 *execPrivate5()()
-66 *execPrivate6()()

--- module creation ---

-72 InitCode(startClass,version)(d0/d1)
-78 InitStruct(initTable,memory,size)(a1/a2,d0)
-84 MakeLibrary(funcInit,structInit,libInit,dataSize,segList)(a0/a1/a2,d0/d1)
-90 MakeFunctions(target,functionArray,funcDispBase)(a0/a1/a2)
-96 FindResident(name)(a1)
-102 InitResident(resident,segList)(a1,d1)

--- diagnostics ---

-108 Alert(alertNum)(d7)
-114 Debug(flags)(d0)

--- interrupts ---

-120 Disable()()
-126 Enable()()
-132 Forbid()()
-138 Permit()()
-144 SetSR(newSR,mask)(d0/d1)
-150 SuperState()()

-156 UserState(sysStack)(d0)
-162 SetIntVector(intNumber,interrupt)(d0/a1)
-168 AddIntServer(intNumber,interrupt)(d0/a1)
-174 RemIntServer(intNumber,interrupt)(d0/a1)
-180 Cause(interrupt)(a1)

--- memory allocation ---

-186 Allocate(freeList,byteSize)(a0,d0)
-192 Deallocate(freeList,memoryBlock,byteSize)(a0/a1,d0)
-198 AllocMem(byteSize,requirements)(d0/d1)
-204 AllocAbs(byteSize,location)(d0/a1)
-210 FreeMem(memoryBlock,byteSize)(a1,d0)
-216 AvailMem(requirements)(d1)
-222 AllocEntry(entry)(a0)
-228 FreeEntry(entry)(a0)

--- lists ---

-234 Insert(list,node,pred)(a0/a1/a2)
-240 AddHead(list,node)(a0/a1)
-246 AddTail(list,node)(a0/a1)
-252 Remove(node)(a1)
-258 RemHead(list)(a0)
-264 RemTail(list)(a0)
-270 Enqueue(list,node)(a0/a1)
-276 FindName(list,name)(a0/a1)

--- tasks ---

-282 AddTask(task,initPC,finalPC)(a1/a2/a3)
-288 RemTask(task)(a1)
-294 FindTask(name)(a1)
-300 SetTaskPri(task,priority)(a1,d0)
-306 SetSignal(newSignals,signalSet)(d0/di)
-312 SetExcept(newSignals,signalSet)(d0/d1)
-318 Wait(signalSet)(d0)
-324 Signal(task,signalSet)(a1,d0)
-330 AllocSignal(signalNum)(d0)
-336 FreeSignal(signalNum)(d0)
-342 AllocTrap(trapNum)(d0)
-348 FreeTrap(trapNum)(d0)

--- messages ---

-354 AddPort(port)(a1)
-360 RemPort(port)(a1)
-366 PutMsg(port,message)(a0/a1)
-372 GetMsg(port)(a0)
-378 ReplyMsg(message)(a1)
-384 WaitPort(port)(a0)
-390 FindPort(name)(a1)

--- libraries ---

-396 AddLibrary(library)(a1)
-402 RemLibrary(library)(a1)
-408 OldOpenLibrary(libName)(a1)
-414 CloseLibrary(library)(a1)
-420 SetFunction(library,funcOffset,newFunction)(a1,a0,d0)
-426 SumLibrary(library)(a1)

--- devices ---

-432 AddDevice(device)(a1)
-438 RemDevice(device)(a1)
-444 OpenDevice(devName,unit,ioRequest,flags)(a0,d0/a1,d1)
-450 CloseDevice(ioRequest)(a1)
-456 DoIO(ioRequest)(a1)
-462 SendIO(ioRequest)(a1)
-468 CheckIO(ioRequest)(a1)
-474 WaitIO(ioRequest)(a1)
-480 AbortIO(ioRequest)(a1)

--- resources ---

-486 AddResource(resource)(a1)
-492 RemResource(resource)(a1)
-498 OpenResource(resName)(a1)

--- private diagnostic support ---

-504 *execPrivate7()()
-510 *execPrivate8()()
-516 *execPrivate9()()

--- misc ---

-522 RawDoFmt(formatString,dataStream,putChProc,putChData)(a0/a1/a2/a3)
-528 GetCC()()
-534 TypeOfMem(address)(a1)
-540 Procure(semaport,bidMsg)(a0/a1)
-546 Vacate(semaport)(a0)
-552 OpenLibrary(libName,version)(a1,d0)

*** functions in Release 1.2 or higher ***

--- signal semaphores (note funny registers found in 1.2 or higher) ---

-558 InitSemaphore(sigSem)(a0)
-564 ObtainSemaphore(sigSem)(a0)
-570 ReleaseSemaphore(sigSem)(a0)
-576 AttemptSemaphore(sigSem)(a0)
-582 ObtainSemaphoreList(sigSem)(a0)
-588 ReleaseSemaphoreList(sigSem)(a0)
-594 FindSemaphore(sigSem)(a1)
-600 AddSemaphore(sigSem)(a1)
-606 RemSemaphore(sigSem)(a1)

--- kickmem support ---

-612 SumKickData()()

--- more memory support ---

-618 AddMemList(size,attributes,pri,base,name)(d0/d1/d2/a0/a1)
-624 CopyMem(source,dest,size)(a0/a1,d0)
-630 CopyMemQuick(source,dest,size)(a0/a1,d0)

*** functions in Release 2.0 or higher ***

--- cache ---

-636 CacheClearU()()
-642 CacheClearE(address,length,caches)(a0,d0/d1)
-648 CacheControl(cacheBits,cacheMask)(d0/d1)

--- misc ---

-654 CreateIORequest(port,size)(a0,d0)
-660 DeleteIORequest(iorequest)(a0)
-666 CreateMsgPort()()
-672 DeleteMsgPort(port)(a0)
-678 ObtainSemaphoreShared(sigSem)(a0)

--- even more memory support ---

-684 AllocVec(byteSize,requirements)(d0/d1)
-690 FreeVec(memoryBlock)(a1)
-696 CreatePrivatePool(requirements,puddleSize,puddleThresh)(d0/dl/d2)
-702 DeletePrivatePool(poolHeader) (a0)
-708 AllocPooled(memSize,poolHeader)(d0/a0)
-714 FreePooled(memory,poolHeader)(a1,a0)

--- misc ---

-720 AttemptSemaphoreShared(sigSem)(a0)
-726 ColdReboot()()
-732 StackSwap (newStack)(a0)

--- task trees ---

-738 ChildFree(tid)(d0)
-744 ChildOrphan(tid)(d0)
-750 ChildStatus(tid)(d0)
-756 ChildWait(tid)(d0)

--- future expansion ---

-762 CachePreDMA(address,length,flags)(a0/a1,d1)
-768 CachePostDMA(address,length,flags)(a0/a1,d1)
-774 *execPrivate10()()

-780 *execPrivate11()()
-786 *execPrivate12()()
-792 *execPrivate13()()

DOS

-30 Open(name,accessMode)(d1/d2)
-36 Close(file)(d1)
-42 Read(file,buffer,length)(d1/d2/d3)
-48 Write(file,buffer,length)(dl/d2/d3)
-54 Input()()
-60 Output()()
-66 Seek(file,position,offset)(d1/dd3)
-72 DeleteFile(name)(d1)
-78 Rename(oldName,newName)(d1/d2)
-84 Lock(name,type)(d1/d2)
-90 UnLock(lock)(d1)
-96 DupLock(lock)(d1)
-102 Examine(lock,fileInfoBlock)(d1/d2)
-108 ExNext(lock,fileInfoBlock)(d1/d2)
-114 Info(lock,parameterBlock)(d1/d2)
-120 CreateDir(name)(d1)
-126 CurrentDir(lock)(d1)
-132 IoErr()()
-138 CreateProc(name,pri,segList,stackSize)(d1/d2/d3/d4)
-144 Exit(returnCode)(d1)
-150 LoadSeg(name)(d1)
-156 UnLoadSeg(seglist)(d1)
-162 *dosPrivate!()()
-168 *dosPrivate2()()
-174 DeviceProc(name)(d1)
-180 SetComment(name,comment)(d1/d2)
-186 SetProtection(name,protect)(d1/d2)
-192 DateStamp(date)(d1)
-198 Delay(timeout)(d1)
-204 WaitForChar(file,timeout)(d1/d2)
-210 ParentDir(lock)(d1)
-216 IsInteractive(file)(d1)
-222 Execute(string,file,file2)(d1/d2/d3)

*** functions in Release 2.0 or higher ***

--- DOS Object creation/deletion ---

-228 AllocDosObject(type,tags)(d1/d2)
-234 FreeDosObject(type,ptr)(d1/d2)

--- Packet Level routines ---

-240 DoPkt(port,action,arg1,arg2,arg3,arg4,arg5)(d1/d2/d3/d4/d5/d6/d7)
-246 SendPkt(dp,port,replyport)(d1/d2/d3)
-252 WaitPkt()()
-258 ReplyPkt(dp,resl,res2)(dl/d2/d3)

-264 AbortPkt(port,pkt)(d1/d2)

--- Record Locking---

-270 LockRecord(fh,offset,length,mode,timeout)(d1/d2/d3/d4/d5)
-276 LockRecords(recArray,timeout)(d1/d2)
-282 UnLockRecord(fh,offset,1ength)(d1/d2/d3)
-288 UnLockRecords(recArray)(d1)

--- Buffered File I/O ---

-294 SelectInput(fh)(d1)
-300 SelectOutput(fh)(d1)
-306 FGetC(fh)(d1)
-312 FPutC(fh,ch)(d1/d2)
-318 UnGetC(fh,character)(d1/d2)
-324 FRead(fh,block,blocklen,number)(d1/d2/d3/d4)
-330 FWrite(fh ,block, blocklen,number)(d1/d2/d3/d4)
-336 FGets(fh,buf,buflen)(d1/d2/d3)
-342 FPuts(fh,str)(d1/d2)
-348 VFWritef(fh,format,argarray)(d1/d2/d3)
-354 VFPrintf(fh,format,argarray)(d1/d2/d3)
-360 Flush(fh)(d1)
-366 SetVBuf(fh,buff,type,size)(d1/d2/d3/d4)

--- DOS Object Management ---

-372 DupLockFromFH(fh)(d1)
-378 OpenFromLock(lock)(d1)
-384 ParentOfFH(fh)(d1)
-390 ExamineFH(fh,fib)(dl/d2)
-396 SetFileDate(name,date)(d1/d2)
-402 NameFromLock(lock,buffer,len)(dl/d2/d3)
-408 NameFromFH(fh,buffer,len)(d1/d2/d3)
-414 SplitName(name,seperator,buf,oldpos,size)(d1/d2/d3/d4/d5)
-420 SameLock(lockl,lock2)(d1/d2)
-426 SetMode(fh,mode)(d1/d2)
-432 ExAll(lock,buffer,size,data,contro1)(d1/d2/d3/d4/d5)
-438 ReadLink(port,lock,path,buffer,size)(d1/d2/d3/d4/d5)
-444 MakeLink(name,dest,soft)(d1/d2/d3)
-450 ChangeMode(type,fh,newmode)(d1/d2/d3)
-456 SetFileSize(fh,pos,mode)(d1/d2/d3)

--- Error Handling ---

-462 SetIoErr(result)(d1)
-468 Fault(code,header,buffer,len)(d1/d2/d3/d4)
-474 PrintFault(code,header)(d1/d2)
-480 ErrorReport(code,type,arg1,device)(d1/d2/d3/d4)
-486 RESERVED

--- Process Management —

-492 Cli()()

-498 CreateNewProc(tags)(d1)
-504 RunCommand(seg,stack,paramptr,paramlen)(d1/d2/d3/d4)
-510 GetConsoleTask()()
-516 SetConsoleTask(task)(d1)
-522 GetFileSysTask()()
-528 SetFileSysTask(task)(d1)
-534 GetArgStr()()
-540 SetArgStr(string)(d1)
-546 FindCliProc(num)(d1)
-552 MaxCli()()
-558 SetCurrentDirName(name)(di)
-564 GetCurrentDirName(buf,len)(d1/d2)
-570 SetProgramName(name)(d1)
-576 GetProgramName(buf,len)(d1/d2)
-582 SetPrompt(name)(d1)
-588 GetPrompt(buf,len)(dl/d2)
-594 SetProgramDir(lock)(d1)
-600 GetProgramDir()()

--- Device List Management ---

-606 SystemTagList(command,tags)(d1/d2)
-612 AssignLock(name,lock)(d1/d2)
-618 AssignLate(name,path)(dl/d2)
-624 AssignPath(name,path)(d1/d2)
-630 AssignAdd(name,lock)(d1/d2)
-636 RemAssignList(name,lock)(d1/d2)
-642 GetDeviceProc(name,dp)(d1/d2)
-648 FreeDeviceProc(dp)(d1)
-654 LockDosList(flags)(d1)
-660 UnLockDosList(flags)(d1)
-666 AttemptLockDosList(flags)(d1)
-672 RemDosEntry(dlist)(d1)
-678 AdDosEntry(dlist)(d1)
-684 FinDosEntry(dlist,name,flags)(d1/d2/d3)
-690 NextDosEntry(dlist,flags)(dl/d2)
-696 MakeDosEntry(name,type)(d1/d2)
-702 FreeDosEntry(dlist)(d1)
-708 IsFileSystem(name)(d1)

---Handler Interface---

-714 Format(filesystem,volumename,dostype)(d1/d2/d3)
-720 Relabel(drive,newname)(d1/d2)
-726 Inhibit(name,onoff)(d1/d2)
-732 AddBuffers(name,number)(d1/d2)

--- Date, Time Routines ---

-738 CompareDates(date1,date2)(d1/d2)
-744 DateToStr(datetime)(d1)
-750 StrToDate(datetime)(d1)

--- Image Management ---

-756 InternalLoadSeg(fh,table,funcarray,stack)(d0/a0/a1/a2)
-762 InternalUnLoadSeg(seglist,freefunc)(d1/a1)
-768 NewLoadSeg(file,tags)(d1/d2)
-774 AddSegment(name,seg,system) (d1/d2/d3)
-780 FindSegment(name,seg,system)(d1/d2/d3)
-786 RemSegment(seg)(d1)

--- Command Support ---

-792 CheckSignal(mask)(d1)
-798 ReadArgs(template,array,args)(d1/d2/d3)
-804 FindArg(keyword,template)(d1/d2)
-810 ReadItem(name,maxchars,cSource)(d1/d2/d3)
-816 StrToLong(string,value)(d1/d2)
-822 MatchFirst(pat,anchor)(d1/d2)
-828 MatchNext(anchor)(d1)
-834 MatchEnd(anchor)(d1)
-840 ParsePattern(pat,buf,buflen)(d1/d2/d3)
-846 MatchPattern(pat,str)(d1/d2)
-852 * Not currently implemented.
-858 FreeArgs(args)(d1)
-864 *--- (1 function slot reserved here) ---
-870 FilePart(path)(d1)
-876 PathPart(path)(d1)
-882 AddPart(dirname,filename,size)(d1/d2/d3)

--- Notification ---

-888 StartNotify(notify)(d1)
-894 EndNotify(notify)(d1)

--- Environment Variable functions---

-900 SetVar(name,buffer,size,flags)(d1/d2/d3/d4)
-906 GetVar(name,buffer,size,flags)(d1/d2/d3/d4)
-912 DeleteVar(name,flags)(d1/d2)
-918 FindVar(name,type)(d1/d2)
-924 *dosPrivate4()()
-930 CliInitNewcli(dp)(a0)
-936 CliInitRun(dp)(a0)
-942 WriteChars(buf,bufLen)(d1/d2)
-948 PutStr(str)(d1)
-954 VPrintf(format,argarray)(d1/d2)
-960 *--- (1 function slot reserved here) ---
-966 ParsePatternNoCase(pat,buf,buflen)(d1/d2/d3)
-972 MatchPatternNoCase(pat,str)(d1/d2)
-978 dosPrivate5()()
-984 SameDevice(lockl,lock2)(d1/d2)

GRAPHICS

-30 BltBitMap (srcBitMap,xSrc,ySrc,destBitMap,xDest,yDest,xSize,ySize,minterm,mask,tempA)

(a0,d0/d1/a1,d2/d3/d4/d5/d6/d7/a2)

-36 BltTemplate (source,xSrc,srcMod,destRP,xDest,yDest,xSize,ySize)(a0,d0/d1/a1,d2/d3/d4/d5)

--- Text routines ---

-42 ClearEOL(rp)(a1)
-48 ClearScreen(rp)(a1)
-54 TextLength(rp,string,count)(a1,a0,d0)
-60 Text(rp,string,count)(a1,a0,d0)
-66 SetFont(rp,textFont)(a1,a0)
-72 OpenFont(textAttr)(a0)
-78 CloseFont(textFont)(a1)
-84 AskSoftStyle(rp)(a1)
-90 SetSoftStyle(rp,style,enable)(a1,d0/d1)

--- Gels routines ---

-96 AddBob(bob,rp)(a0/a1)
-102 AddVSprite(vSprite,rp)(a0/a1)
-108 DoCollision(rp)(a1)
-114 DrawGList(rp,vp)(a1,a0)
-120 InitGels(head,tail,gelsInfo)(a0/a1/a2)
-126 InitMasks(vSprite)(a0)
-132 RemIBob(bob,rp,vp)(a0/a1/a2)
-138 RemVSprite(vSprite)(a0)
-144 SetCollision(num,routine,gelsInfo)(d0/a0/a1)
-150 SortGList(rp)(a1)
-156 AddAnimOb(anOb,anKey,rp)(a0/a1/a2)
-162 Animate(anKey,rp)(a0/a1)
-168 GetGBuffers(anOb,rp,flag)(a0/a1,d0)
-174 InitGMasks(anOb)(a0)

--- General graphics routines ---

-180 DrawEllipse(rp,xCenter,yCenter,a,b)(a1,d0/d1/d2/d3)
-186 AreaEllipse(rp,xCenter,yCenter,a,b)(a1,d0/d1/d2/d3)
-192 LoadRGB4(vp,colors,count)(a0/a1,d0)
-198 InitRastPort(rp)(a1)
-204 InitVPort(vp)(a0)
-210 MrgCop(view)(a1)
-216 MakeVPort(view,vp)(a0/a1)
-222 LoadView(view)(a1)
-228 WaitBlit()()
-234 SetRast(rp,pen)(a1,d0)
-240 Move(rp,x,y)(a1,d0/d1)
-246 Draw(rp,x,y)(a1,d0/d1)
-252 AreaMove(rp,x,y)(a1,d0/d1)
-258 AreaDraw(rp,x,y)(a1,d0/d1)
-264 AreaEnd(rp)(a1)
-270 WaitTOF()()
-276 QBlit(blit)(a1)
-282 InitArea(areaInfo,vectorBuffer,maxVectors)(a0/a1,d0)
-288 SetRGB4(vp,index,red,green,blue)(a0,d0/d1/d2/d3)

-294 QBSBlit(blit)(a1)
-300 BitClear(memBlock,byteCount,f1ags)(a1,d0/d 1)
-306 RectFill(rp,xMin,yMin,xMax,yMax)(a1,d0/d1/d2/d3)
-312 BltPattern(rp,mask,xMin,yMin,xMax,yMax,maskBPR)(a1,a0,d0/d1/d2/d3/d4)
-318 ReadPixel(rp,x,y)(a1,d0/d1)
-324 WritePixel(rp,x,y)(a1,d0/d1)
-330 Flood(rp,mode,x,y)(a1,d2,d0/d1)
-336 PolyDraw(rp,count,polyTable)(a1,d0/a0)
-342 SetAPen(rp,pen)(a1,d0)
-348 SetBPen(rp,pen)(a1,d0)
-354 SetDrMd(rp,drawMode)(a1,d0)
-360 InitView(view)(a1)
-366 CBump(copList)(a1)
-372 CMove(copList,destination,data)(a1,d0/d 1)
-378 CWait(copList,v,h)(a1,d0/d1)
-384 VBeamPos()()
-390 InitBitMap(bitMap,depth,width,height)(a0,d0/d1/d2)
-396 ScrollRaster(rp,dx,dy,xMin,yMin,xMax,yMax)(a1,d0/d1/d2/d3/d4/d5)
-402 WaitBOVP(vp)(a0)
-408 GetSprite(sprite,num)(a0,d0)
-414 FreeSprite(num)(d0)
-420 ChangeSprite(vp,sprite,newData)(a0/a1/a2)
-426 MoveSprite(vp,sprite,x,y)(a0/a1,d0/d 1)
-432 LockLayerRom(layer)(a5)
-438 UnlockLayerRom(layer)(a5)
-444 SyncSBitMap(layer)(a0)
-450 CopySBitMap(layer)(a0)
-456 OwnBlitter()()
-462 DisownBlitter()()
-468 InitTmpRas(tmpRas,buffer,size)(a0/a1,d0)
-474 AskFont(rp,textAttr)(a1,a0)
-480 AddFont(textFont)(a1)
-486 RemFont(textFont)(a1)
-492 AllocRaster(width,height)(d0/d1)
-498 FreeRaster(p,width,height)(a0,d0/d1)
-504 AndRectRegion(region,rectangle)(a0/a1)
-510 OrRectRegion(region,rectangle)(a0/a1)
-516 NewRegion()()
-522 ClearRectRegion(region,rectangle)(a0/a1)
-528 ClearRegion(region)(a0)
-534 DisposeRegion(region)(a0)
-540 FreeVPortCopLists(vp)(a0)
-546 FreeCopList(copList)(a0)
-552 CLipBlit (srcRP,xSrc,ySrc,destRP,xDest,yDest,xSize,ySize,minterm) (a0,d0/d1/a1,d2/d3/d4/d5/d6)
-558 XorRectRegion(region,rectangle)(a0/a1)
-564 FreeCprList(cprList)(a0)
-570 GetColorMap(entries)(d0)
-576 FreeColorMap(colorMap)(a0)
-582 GetRGB4(colorMap,entry)(a0,d0)
-588 ScrollVPort(vp)(a0)
-594 UCopperListInit(uCopList,n)(a0,d0)
-600 FreeGBuffers(anOb,rp,flag)(a0/a1,d0)
-606 BLtBitMapRastPort (srcBM,x,y,destRP,x,y,Wld,Height,minterm)(a0,d0/d1/a1,d2/d3/d4/d5/d6)
-612 OrRegionRegion(srcRegion,destRegion)(a0/a1)

-618 XorRegionRegion(srcRegion,destRegion)(a0/a1)
-624 AndRegionRegion(srcRegion,destRegion)(a0/a1)
-630 SetRGB4CM(colorMap,index,red,green,blue)(a0,d0/d1/d2/d3)
-636 BltMaskBitMapRastPort
(srcBM,x,y,destRP,x,y,Wid,High,mterm,Mask)(a0,d0/d1/a1,d2/d3/d4/d5/d6/a2)
-642 RESERVED
-648 RESERVED
-654 AttemptLockLayerRom(layer)(a5)

*** functions in Release 2.0 or higher ***

-660 GfxNew(gfxNodeType)(d0)
-666 GfxFree(gfxNodePtr)(a0)
-672 GfxAssociate(associateNode,gfxNodePtr)(a0/a1)
-678 BitMapScale(bitScaleArgs)(a0)
-684 ScalerDiv(factor,numerator,denominator)(d0/d1/d2)
-690 TextFit
(rp,string,strLen,textExtent,constrainingExtent,strDirection,constrainingBitWidth,constrainingBitHeight)
(a1,a0,d0/a2)

INTUITION

-30 OpenIntuition()()
-36 Intuition(iEvent)(a0)
-42 AddGadget(window,gadget,position)(a0/a1,d0)
-48 ClearDMRequest(window)(a0)
-54 ClearMenuStrip(window)(a0)
-60 ClearPointer(window)(a0)
-66 CloseScreen(screen)(a0)
-72 CloseWindow(window)(a0)
-78 CloseWorkBench()()
-84 CurrentTime(seconds,micros)(a0/a1)
-90 DisplayAlert(alertNumber,string,height)(d0/a0,dl)
-96 DisplayBeep(screen)(a0)
-102 DoubleClick(sSeconds,sMicros,cSeconds,cMicros)(d0/d1/d2/d3)
-108 DrawBorder(rp,border,leftOffset,topOffset)(a0/a1,d0/d1)
-114 DrawImage(rp,image,leftOffset,topOffset)(a0/a1,d0/d1)
-120 EndRequest(requester,window)(a0/a1)
-126 GetDefPrefs(preferences,size)(a0,d0)
-132 GetPrefs(preferences,size)(a0,d0)
-138 InitRequester(requester)(a0)
-144 ItemAddress(menuStrip,menuNumber)(a0,d0)
-150 ModifyIDCMP(window,flags)(a0,d0)
-156 ModifyProp (gadget,window,requester,flags,horizPot,vertPot,horizBody,vertBody)
(a0/a1/a2,d0/d1/d2/d3/d4)
-162 MoveScreen(screen,dx,dy)(a0,d0/d1)
-168 MoveWindow(window,dx,dy)(a0,d0/d1)
-174 OffGadget(gadget,window,requester)(a0/a1/a2)
-180 OffMenu(window,menuNumber)(a0,d0)
-186 OnGadget(gadget,window,requester)(a0/a1/a2)
-192 OnMenu(window,menuNumber)(a0,d0)
-198 OpenScreen(newScreen)(a0)
-204 OpenWindow(newWindow)(a0)

-210 OpenWorkBench()()
-216 PrintIText(rp,iText,left,top)(a0/a1,d0/d1)
-222 RefreshGadgets(gadgets,window,requester)(a0/a1/a2)
-228 RemoveGadget(window,gadget)(a0/a1)
-234 ReportMouse(flag,window)(d0/a0)
-240 Request(requester,window)(a0/a1)
-246 ScreenToBack(screen)(a0)
-252 ScreenToFront(screen)(a0)
-258 SetDMRequest(window,requester)(a0/a1)
-264 SetMenuStrip(window,menu)(a0/a1)
-270 SetPointer(window,pointer,height,width,xOffset,yOffset)(a0/a1,d0/d1/d2/d3)
-276 SetWindowTitles(window,windowTitle,screenTitle)(a0/a1/a2)
-282 ShowTitle(screen,showIt)(a0,d0)
-288 SizeWindow(window,dx,dy)(a0,d0/d1)
-294 ViewAddress()()
-300 ViewPortAddress(window)(a0)
-306 WindowToBack(window)(a0)
-312 WindowToFront(window)(a0)
-318 WindowLimits(window,widthMin,heightMin,widthMax,heightMax)(a0,d0/d1/d2/d3)
-324 SetPrefs(preferences,size,inform)(a0,d0/d1)
-330 IntuiTextLength(iText)(a0)
-336 WBenchToBack()()
-342 WBenchToFront()()
-348 AutoRequest (window,body,posText,negText,pFlag,nFlag,width,height)(a0/a1/a2/a3,d0/d1/d2/d3)
-354 BeginRefresh(window)(a0)
-360 BuildSysRequest (window,body,posText,negText,flags,width,height)(a0/a1/a2/a3,d0/d1/d2)
-366 EndRefresh(window,complete)(a0,d0)
-372 FreeSysRequest(window)(a0)
-378 MakeScreen(screen)(a0)
-384 RemakeDisplay() ()
-390 RethinkDisplay()()
-396 AllocRemember(rememberKey,size,flags)(a0,d0/d1)
-402 AlohaWorkbench(wbport)(a0)
-408 FreeRemember(rememberKey,reallyForget)(a0,d0)
-414 LockIBase(dontknow)(d0)
-420 UnlockIBase(ibLock)(a0)

*** functions in Release 1 .2 or higher ***

-426 GetScreenData(buffer,size,type,screen)(a0,d0/d1/a1)
-432 RefreshGList(gadgets,window,requester,numGad)(a0/a1/a2,d0)
-438 AddGList(window,gadget,position,numGad,requester)(a0/a1,d0/d1/a2)
-444 RemoveGList(remPtr,gadget,numGad)(a0/a1,d0)
-450 ActivateWindow(window)(a0)
-456 RefreshWindowFrame(window)(a0)
-462 ActivateGadget(gadgets,window,requester) (a0/a1/a2)
-468 NewModifyProp (gadget,window,requester,flags,horizPot,vertPot,horizBody,vertBody,numGad)
(a0/a1/a2,d0/d1/d2/d3/d4/d5)

*** functions in Release 2.0 or higher ***

-474 QueryOverscan(displayID,rect,oScanType)(a0/a1,d0)
-480 MoveWindowInFrontOf(window,behindWindow)(a0/a1)
-486 ChangeWindowBox(window,left,top,width,height)(a0,d0/d1/d2/d3)

-492 SetEditHook(hook)(a0)
-498 SetMouseQueue(window,queueLength)(a0,d0)
-504 ZipWindow(window)(a0)

--- public screens ---

-510 LockPubScreen(name)(a0)
-516 UnlockPubScreen(name,screen)(a0/a1)
-522 LockPubScreenList()()
-528 UnlockPubScreenList()()
-534 NextPubScreen(screen,namebuf)(a0/a1)
-540 SetDefaultPubScreen(name)(a0)
-546 SetPubScreenModes(modes)(d0)
-552 PubScreenStatus(screen,statusFlags)(a0,d0)
-558 ObtainGIRPort(gInfo)(a0)
-564 ReleaseGIRPort(rp)(a0)
-570 GadgetMouse(gadget,gInfo,mousePoint)(a0/a1/a2)
-576 *intuitionPrivate1()()
-582 GetDefaultPubScreen(nameBuffer)(a0)
-588 EasyRequestArgs(window,easyStruct,idcmpPtr,args)(a0/a1/a2/a3)
-594 BuildEasyRequestArgs(window,easyStruct,idcmp,args)(a0/a1,d0/a3)
-600 SysReqHandler(window,idcmpPtr,waitInput)(a0/a1,d0)
-606 OpenWindowTagList(newWindow,tagList)(a0/a1)
-612 OpenScreenTagList(newScreen,tagList)(a0/a1)

--- new Image functions --- .

-618 DrawImageState(rp,image,leftOffset,topOffset,state,drawInfo)(a0/a1,d0/d1/d2/a2)
-624 PointInImage(point,image)(d0/a0)
-630 EraseImage(rp,image,leftOffset,topOffset)(a0/a1,d0/d1)
-636 NewObjectA(classPtr,classID,tagList)(a0/a1/a2)
-642 DisposeObject(object)(a0)
-648 SetAttrsA(object,tagList)(a0/a1)
-654 GetAttr(attrID,object,storagePtr)(d0/a0/a1)

--- special set attribute call for gadgets ---

-660 SetGadgetAttrsA(gadget,window,requester,tagList)(a0/a1/a2/a3)
-666 NextObject(objectPtrPtr)(a0)
-672 *intuitionPrivate2()()
-678 MakeClass (classID,superClassID,superClassPtr,instanceSize,flags)(a0/a1/a2,d0/d1)
-684 AddClass(classPtr)(a0)
-690 GetScreenDrawInfo(screen)(a0)
-696 FreeScreenDrawInfo(screen,drawInfo)(a0/a1)
-702 ResetMenuStrip(window,menu)(a0/a1)
-708 RemoveClass(classPtr)(a0)
-714 FreeClass(classPtr)(a0)
-720 *intuitionPrivate3()()
-726 *intuitionPrivate4()()

DISKFONT

-30 OpenDiskFont(textAttr)(a0)

-36 AvailFonts(buffer,bufBytes,flags)(a0,d0/d1)

*** functions in Release 1.2 or higher ***

-42 NewFontContents(fontsLock,fontName)(a0/a1)
-48 DisposeFontContents(fontContentsHeader)(a1)

*** functions in Release 2.0 or higher ***

-54 NewScaleDiskFont(sourceFont,destTextAttr)(a0/a1)

Apéndice 2: Referencia técnica de los principales registros de hardware

A continuación se explican las direcciones de memoria o registros más importantes relacionadas con el conjunto de chips auxiliares de la arquitectura. Es importante entender que el acceso directo a estos registros dentro del ambiente multiproceso debe ser realizado con cuidado y pleno conocimiento de las características del sistema operativo, pues éste contiene funciones importantes que trabajan sobre estos registros y alguna modificación incoherente a los mismos puede conducir a resultados inesperados.

De cualquier manera, al tomar el control total de la máquina y eliminar el ambiente multiproceso, estos registros pueden ser accedidos sin mayor problema tomando en consideración la documentación aquí presentada.

Es importante recordar que aquí se presentan las características de la primera revisión del hardware, ha habido 2 revisiones en total, pero la última nunca fue totalmente documentada, de cualquier modo, todo lo presentado aquí es compatible con ambas.

Nota: Los comentarios de esta compilación están en el idioma inglés, para conservar las ideas y significados originales y además poder distribuirlos en Internet si así se requiere.

·BitPlane & DisplayControl (*=Enhanced Chip Set only)

The Amiga has great flexibility in displaying graphics at different resolutions and positions on the monitor. The hardware registers associated with the display are nearly always loaded by the copper and not with the 680x0 processor.

```
#BPLCON0=$100
#BPLCON1=$102
#BPLCON2=$104
#BPLCON3=$106      ;(ECS only)
```

BIT#	BPLCON0	BPLCON1	BPLCON2
15	HIRES (70ns pixles)		
14	BPU2 \		
13	BPU1 |#BitPlanes(0-6)		
12	BPU0 /		
11	HOMOD Hold & Modify		
10	DBLPF DualPlayField		
09	COLOR CompositeEnable		
08	GAUD GenlockAudio		
07		PF2H3 \	
06	*SHRES SuperHires	PF2H2 |Playfield2	PF2PRI DBLPF priority
05	*BPLHWRM	PF2H1 |horizontal	PF2P2
04	*SPRHWRM	PF2H0 /scroll	PF2P1 Priority to sprites
03	LPEN LightPenEnable	PF1H3 \	PF2P0
02	LACE Interlace	PF1H2 |Playfield1	PF1P2
01	ERSY ExternalSync	PF1H1 |Horizontal	PF1P1 Priority to sprites
00		PF1H0 /scroll	PF1P0

```
#BPL0PTH=$E0      ;BitPlane Pointer 0 High Word
#BPL0PTL=$E2      ;BitPlane Pointer 0 Low Word
#BPL1PTH=$E4
#BPL1PTL=$E6
```

#BPL2PTH=$E8
#BPL2PTL=$EA
#BPL3PTH=$EC
#BPL3PTL=$EE
#BPL4PTH=$F0
#BPL4PTL=$F2
#BPL5PTH=$F4
#BPL5PTL=$F6

Each pair of registers contain an 18 bit pointer to the address of BitPlanes data in chip memory. They *MUST be reset every frame usually by the copper.*

#BPL1MOD=$108 ;Bitplane Modulo for Odd Planes
#BPL2MOD=$10A ;Bitplane Modulo for Even Planes

At the end of each display line, the BPLxMODs are added to the the BitPlane Pointers so they point to the address of the next line.

#DIWSTOP=$090 ;display window stop
#DIWSTRT=$08E ;display window start

These two registers control the display window size and position. *The following bits are assigned*

```
BIT# 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
     V7 V6 V5 V4 V3 V2 V1 V0 H7 H6 H5 H4 H3 H2 H1 H0
```

For DIWSTRT V8=0 & H8=0 restricting it to the upper left of the screen. For DIWSTOP V8=1 & H8=1 restricting it to the lower right of the screen.

#DDFSTOP=$094 ;data fetch stop
#DDFSTRT=$092 ;data fetch start

The two display data fetch registers control when and how many words are fetched from the bitplane for each line of display.

Typical values are as follows:

lores 320 pixels, DDFSTRT & DDFSTOP = $38 & $D0
hires 640 pixels, DDFSTRT & DDFSTOP = $3C & $d4

If smooth scrolling is enabled DDFSTRT should be 2 less than above.

#BPL1DAT $110 ;BitPlane Data parallel to serial converters
#BPL2DAT $112
#BPL3DAT $114
#BPL4DAT $116
#BPL5DAT $118
#BPL6DAT $11A

These 6 registers receive the DMA data fetched by the BitPlane engine, and output it serially to the Amiga DACS, triggered by writing to BPL1DAT. Not intended for programming access.

·The Copper

The Copper is found on the Agnus chip, it's main job is to 'poke' values into the hardware registers in sync with the video beam. The main registers it updates are BitPlane ptrs, Sprites and other control words that HAVE to be reset every frame. It's also used to split the screen vertically as it is capable of waiting for certain video beam positions before writing data. It's also capable of waiting for the blitter to finish as well as skipping instructions if beam position is equal to certain values.

#COP1LCH=$080
#COP1LCL=$082

#COP2LCH=$084
#COP2LCL=$086

Each pair of registers contain an 18 bit pointer to the address of a Copper List in chip mem. The Copper will automatically jump to the address in COP1 at the beginning of the frame and is able to jump to COP2 *if the following strobe is written to.*

#COPJMP1=$88
#COPJMP2=$8A

When written to these addresses cause the copper to jump to the locations held in COP1LC & COP2LC. The Copper can write to these registers itself causing its own indirect jump.

#COPCON=$2E

By setting bit 1 of this register the copper is allowed to access the blitter hardware.

The copper fetches two words for each instruction from its current copper list. The three instructions it can perform and their relevant bits are as follow:

BIT#	MOVE		WAIT UNTIL		SKIP IF	
15	x	RD15	VP7	BFD	VP7	BFD
14	x	RD14	VP6	VE6	VP6	VE6
13	x	RD13	VP5	VE5	VP5	VE5
12	x	RD12	VP4	VE4	VP4	VE4
11	x	RD11	VP3	VE3	VP3	VE3
10	x	RD10	VP2	VE2	VP2	VE2
09	x	RD09	VP1	VE1	VP1	VE1
08	DA8	RD08	VP0	VE0	VP0	VE0
07	DA7	RD07	HP8	HE8	HP8	HE8
06	DA6	RD06	HP7	HE7	HP7	HE7
05	DA5	RD05	HP6	HE6	HP6	HE6
04	DA4	RD04	HP5	HE5	HP5	HE5
03	DA3	RD03	HP4	HE4	HP4	HE4
02	DA2	RD02	HP3	HE3	HP3	HE3
01	DA1	RD01	HP2	HE2	HP2	HE2
00	0	RD00	1	0	1	1

The MOVE instruction shifts the value held in RD15-0 to the destination address calculated by $DFF000 +DA8-1.

The WAIT UNTIL instruction places the copper in a wait state until the video beam position is past HP,VP

(xy coordinates). The Copper first logical ANDS (masks) the video beam with HE,VE before doing the comparison. If BFD is set then the blitter must also be finished before the copper will exit its wait state.

The SKIP IF instruction is similar to the WAIT UNTIL instruction but instead of placing the copper in a wait state if the video beam position fails the comparison test it skips the next MOVE instruction.

·Colour Registers

The following 32 color registers can each represent one of 4096 colors.

#COLOR00=$180	#COLOR08=$190	#COLOR16=$1A0	#COLOR24=$1B0
#COLOR01=$182	#COLOR09=$192	#COLOR17=$1A2	#COLOR25=$1B2
#COLOR02=$184	#COLOR10=$194	#COLOR18=$1A4	#COLOR26=$1B4
#COLOR03=$186	#COLOR11=$196	#COLOR19=$1A6	#COLOR27=$1B6
#COLOR04=$188	#COLOR12=$198	#COLOR20=$1A8	#COLOR28=$1B8
#COLOR05=$18A	#COLOR13=$19A	#COLOR21=$1AA	#COLOR29=$1BA
#COLOR06=$18C	#COLOR14=$19C	#COLOR22=$1AC	#COLOR30=$1BC
#COLOR07=$18E	#COLOR15=$19E	#COLOR23=$1AE	#COLOR31=$1BE

The bit usage for each of the 32 colors is:

```
BIT# 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
      x  x  x  x  R3 R2 R1 R0 G3 G2 G1 G0 B3 B2 B1 B0
```

This represents a combination of 16 shades of red, green and blue.

·Blitter Control

The Blitter is located on the Agnus chip, it's main function is to move blocks of data around chip mem. It has 3 input channels A,B & C and 1 output channel D. A simple block move would use 1 input channel and 1 output channel, taking 4 clock ticks per cycle. A complex move such as a moving a shape to a destination with a cookie cut would use all 3 input channels and the output channel taking 8 clock ticks per cycle.

The main parameters of the blitter include the width and height of the block to be moved (width is in multiples of words), a start address for each channel, a modulo for each channel that is added to the address at the end of each Line so they point to the next line, a logic function that specifies which input channels data will be sent to the destination channel.

·Logic Function Calculation.

The following is a table to work out the logic function (known as the minterm) for a blitter operation.

A	B	C	D
0	0	0	LF0
0	0	1	LF1
0	1	0	LF2
0	1	1	LF3
1	0	0	LF4
1	0	1	LF5
1	1	0	LF6
1	1	1	LF7

If the Blitter is set up so that channel A points to the cookie, B points to the shape to be copied and C&D point to the destination bitplane (such as how Blitz 2 uses the blitter) we would specify the following conditions:

When A is 1 then make D=B
When A is 0 then make D=C

Using the above table we calculate the values of LF0-LF7 when these two conditions are met. The top line has A=0 so LF0 becomes the value in the C column which is a 0. A is 0 in the first 4 rows so LF0-LF3 all reflect the bits in the C column (0101) and A=1 in the lower 4 rows so LF4-LF7 reflect the bits in the B column (0011).

This generates a minterm LF0-LF7 of %10101100 or in hex $AC.

Note: read the values of LF7 to LF0 from bottom to top to calculate the correct hexadecimal minterm.

```
#BLTAPTH=$50
#BLTAPTL=$52

#BLTBPTH=$4C
#BLTBPTL=$4E

#BLTCPTH=$48
#BLTCPTL=$4A

#BLTDPTH=$54
#BLTDPTL=$56
```

Each pair of registers contain an 18 bit pointer to the start address of the 4 blitter channels in chip mem.

```
#BLTAMOD=$64
#BLTBMOD=$62
#BLTCMOD=$60
#BLTDMOD=$66
```

The 4 modulo values are added to the blitter pointers at the end of each line.

```
#BLTADAT=$74
#BLTBDAT=$72
#BLTCDAT=$70
```

If a blitter channel is disabled the BLTxDAT register can be loaded with a constant value which will remain unchanged during the blit operation.

```
#BLTAFWM=$44      ;Blitter first word mask for source A
#BLTALWM=$46      ;Blitter last word mask for source A
```

During a Blitter operation these two registers are used to mask the contents of BLTADAT for the first and last word of every line.

```
#BLTCON0=$100
#BLTCON1=$102
```

The following bits in BLTCON0 & BLTCON1 are as follows.

BIT#	BLTCON0	BLTCON1
15	ASH3	BSH3
14	ASH2	BSH2
13	ASH1	BSH1
12	ASH0	BSH0
11	USEA	x
10	USEB	x
09	USEC	x
08	USED	x
07	LF7	x
06	LF6	x
05	LF5	x
04	LF4	EFE
03	LF3	IFE
02	LF2	FCI
01	LFI	DESC
00	LF0	0 (1 =line mode)

ASH is the amount that source A is shifted (barrel rolled)
USEx enables each of the 4 blitter channels
LF holds the logic function as discussed previously in this section
BSH is the amount that source B is shifted (barrel rolled)
EFE is the Exclusive Fill Enable flag
IFE is the Inclusive Fill Enable flag
FCI is the Fill Carry Input
DESC is the descending flag (blitter uses decreasing addressing)

#BLTSIZE=$58

By writing the height and width of the blit operation to BLTSIZE the the blitter will start the operation. Maximum size is 1024 high and 64 words (1024 bits) wide. The following defines bits in BLITZSIZE

```
BIT# 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
     h9 h8 h7 h6 h5 h4 h3 h2 h1 h0 w5 w4 w3 w2 w1 w0
```

```
#BLTSIZV= $5C        ;(ECS ONLY)
#BLTSIZH =$5E        ;(ECS ONLY)
```

With the new ECS writing to BLTSIZV first and then BLTSZH the blitter can operate on blocks as large as 32K x 32K pixels in size.

The Blitter is also able to perform linedrawing, filled polygon and other functions. Details in this topic are beyond the scope of this document, please consult the Amiga Hardware Reference Guides.

·Audio Control

The Amiga has 4 channels of 8 bit audio, each with their own memory access, period and 6 bit volume control (that permits also real stereo audio of 14 bits). The following are a list of the applicable hardware registers.

```
#AUD0LCH=$A0        ;pairs of 24 bit memory pointers to audio data in chip mem
#AUD0LCL=$A2
#AUD1LCH=$B0
#AUD1LCL=$B2
#AUD2LCH=$C0
#AUD2LCL=$C2
#AUD3LCH=$D0
#AUD3LCL=$D2

#AUD0LEN=$A4
#AUD1LEN=$B4
#AUD2LEN=$C4
#AUD3LEN=$D4

#AUD0PER=$A6        ;period
#AUD1PER=$B6
#AUD2PER=$C6
#AUD3PER=$D6

#AUD0VOL=$A8        ;volume registers (0-63)
#AUD1VOL=$B8
#AUD2VOL=$C8
#AUD3VOL=$D8

#AUD0DAT=$AA
#AUD1DAT=$BA
#AUD2DAT=$CA
#AUD3DAT=$DA
```

·Sprite Control

The Amiga hardware is capable of displaying eight 4 colour sprites or four 16 colour sprites. Standard control of sprites is done by using the copper to setup the 8 sprite pointers at the beginning of each *frame*.

```
#SPR0PTH=$120       ;pairs of 24 bit memory pointers to sprite data in chip mem
#SPR0PTL=$122
#SPR1PTH=$124
#SPR1PTL=$126
#SPR2PTH=$128
#SPR2PTL=$12A
#SPR3PTH=$12C
#SPR3PTL=$12E
#SPR4PTH=$130
#SPR4PTL=$132
#SPR5PTH=$134
#SPR5PTL=$136
#SPR6PTH=$138
#SPR6PTL=$13A
#SPR7PTH=$13C
#SPR7PTL=$13E
```

The pointers should point to data that is begins with two words containing the SPRPOS & SPRCTL

values for that sprite, followed by its image data and with two null words that terminate the data.

```
#SPR0POS=$140    #SPR0CTL=$142    #SPR0DATA=$144    #SPR0DATB=$146
#SPR1POS=$148    #SPR1CTL=$14A    #SPR1DATA=$14C    #SPR1DATB=$14E
#SPR2POS=$150    #SPR2CTL=$152    #SPR2DATA=$154    #SPR2DATB=$156
#SPR3POS=$158    #SPR3CTL=$15A    #SPR3DATA=$15C    #SPR3DATB=$15E
#SPR4POS=$160    #SPR4CTL=$162    #SPR4DATA=$164    #SPR4DATB=$166
#SPR5POS=$168    #SPR5CTL=$16A    #SPR5DATA=$16C    #SPR5DATB=$16E
#SPR6POS=$170    #SPR6CTL=$172    #SPR6DATA=$174    #SPR6DATB=$176
#SPR7POS=$178    #SPR7CTL=$17A    #SPR7DATA=$17C    #SPR7DATB=$17E
```

Using standard sprite DMA the above registers are all loaded from the sprite data pointed to in chip mem by the sprite pointers. These registers are only of interest to people wanting to 'multiplex' sprites by using the copper to load these registers rather than sprite DMA.

The following is bit definitions of both SPRPOS and SPRCTL.

```
BIT#    15  14  13  12  11  10  09  08  07  06  05  04  03  02  01  00
SPRPOS  SV7 SV6 SV5 SV4 SV3 SV2 SV1 SV0 SH8 SH7 SH6 SH5 SH4 SH3 SH2 SH1
SPRCTL  EV7 EV6 EV5 EV4 EV3 EV2 EV1 EV0 ATT  X   X   X   X  SV8 EV8 SHO
```

SV is the vertical start position of the sprite
SH is the horizontal position of the sprite (calulated in lores pixels only)
EV is the end vertical position
ATT is the sprite attached bit (connects odd sprites to their predecessors)

·Interupt Control

```
#INTENA=$9A      ;interupt enable write address
#INTENAR=$1C     ;interupt enable read address

#INTREQ=$9C      ;interupt request write address
#INTREQR=$1E     ;interupt request read address
```

INTENA is used to enable or disable interupts. If the value written to INTENA has bit 15 set any other of the bits enable their corresponding interupts. If bit 15 is clear any of the other bits set will disable their corresponding interupts.

INTENAR will return which interupts are currently enabled.

INTREQ is used to initiate or clear an interupt. It is mostly used to clear the interupt by the interupt handler. Again bit 15 states whether the corresponding interupts will be requested or cleared.

INTREQR returns which interupts are currently requested.

The following bit definitions relate to the 4 interupt control registers.

Bit Name	Level	Description
15 SET/CLR		determines if bits written with 1 are set or cleared
14 INTEN		master interupt enable
13 EXTER	6	external interupt
12 DSKSYN	5	disk sync register (same as DSKSYNC)
11 RBF	5	serial port Receive Buffer Full
10 AUD3	4	audio channel 3 finished
09 AUD2	4	audio channel 2 finished
08 AUDI	4	audio channel 1 finished
07 AUD0	4	audio channel 0 fin ished
06 BLIT	3	blitter finished
05 VERTB	3	start of vertical blank interupt
04 COPER	3	copper
03 PORTS	2	I/O ports and timers
02 SOFT	1	reserved for software initiated interupts
01 DSKBLK	1	disk block finished
00 TBE	1	serial port Transmit Buffer Empty

The following locations hold the address of the 68000 interupt handler code in memory for each level of interupt.

Level	68000 Address
6	$78
5	$74
4	$70
3	$6c
2	$68
1	$64

·DMA Control

DMA stands for direct memory access. Chip mem can be accessed by the display, blitter, copper, audio, sprites and diskdrive without using the 68000 processor. DMACON enables the user to lock out any of these from having direct memory access (DMA) to chipmem.

As with INTENA bit 15 of DMACON signals whether the write operation should clear or set the relevant bits of the DMA control.

DMACONR will not only return which channels have DMA access but has flags BBUSY which return true if the blitter is in operation and BZERO which return if the Blitter has generated any 1's from it logic function (useful for collision detection etc.)

```
#DMACON=$96      ;DMA control write (clear or set)
#DMACONR=$02     ;DMA control read (and blitter status)
```

The following are the bits assigned to the two DMACON registers.

Bit Name	Description
15 SET/CLR	determines if bits written with 1 are set or cleared
14 BBUSY	blitter busy flag
13 BZERO	blitter logic zero
12 X	
11 X	
10 BLTPRI	"blitter nasty sig". blitter has DMA priority over CPU
09 DMAEN	enable all DMA below
08 BPLEN	BitPlane DMA enable
07 COPEN	Copper DMA enable
06 BLTEN	Blitter DMA enable
05 SPREN	Sprite DMA enable
04 DSKEN	Disk DMA enable
03 AUD3EN	Audio channel 3 DMA enable
02 AUD2EN	Audio channel 2 DMA enable
01 AUD1EN	Audio channel 1 DMA enable
00 AUD0EN	Audio channel 0 DMA enable

·Miscellaneous Amiga Chip Locations

The following is a list of the other $dff000 addresses not covered by the previous sections. Because of their complex nature other texts should be referred to for more information.

```
#ADKCON=$09E          ;Audio/Disk control write
#ADKCONR=$010         ;Audio/Disk control read
#BEAMCON0=$1DC        ;ECS Beam Counter Control Register
#CLXCON=$098          ;Collision control register
#CLXDAT=$00E          ;Collision data register

#DENISEID=$07C        ;ECS Denise chip revsion level
#DIWHIGH=$1E4         ;ECS display window high
#DSKBYTER=$01A        ;disk data byte and status read
#DISKDAT=$026         ;disk DMA data write
#DISKDATR=$008        ;disk DMA data read
#DSKLEN=$024          ;disk length
#DSKPTH=$020          ;disk pointer high
#DSKPTL=$022          ;disk pointer low
#DSKSYNC=$07E         ;disk sync register

#HBSTOP=$1C6          ;ECS horizontal line position for HBLANK stop
#HBSTRT=$1C4          ;ECS horizontal line position for HBLANK start
#HCENTER=$1E2         ;ECS horizontal position for Vsync on interlace
#HSSTOP=$1C2          ;ECS horizontal line position for HSYNC stop
#HSSTRT=$1DE          ;ECS horizontal line position for HSYNC strt
#HTOTAL=$1C0          ;ECS highest number count for horizontal line

#JOY0DAT=$00A         ;joystick mouse data left up/dwn
#JOY1DAT=$00C         ;joystick mouse data right up/dwn
#JOYTEST=$036         ;mouse counters write
```

```
#POT0DAT=$012        ;pot counter data left pair
#POT1DAT=$014        ;pot counter data right pair
#POTGO=$034          ;pot port data write and start
#POTGOR=$016         ;pot port data read
#REFPTR=$028         ;refresh pointer
#SERDAT=$030         ;serial port data write (with stop bit)
#SERDATR=$018        ;serial port data read and status bits
#SERPER=$032         ;baud rate and 9 bit word flag

#STREQU=$038         ;strobe for horizontal sync with VB and EQU
#STRHOR=$03C         ;strobe for horizontal sync
#STRLONG=$03E        ;ECS strobe for id of long horizontal line
#STRVBL=$03A         ;strobe for horizontal sync with VB
#VBSTOP=$1CE         ;ECS vertical line for vblank stop
#VBSTRT=$1CC         ;ECS vertical line for vblank start

#VHPOSR=$006         ;video beam position
#VHPOSW=$02C         ;write vertical beam position
#VPOSR=$004          ;video beam position (vertical most significant bit)
#VPOSW=$02A          ;write vertical beam position MSB

#VSSTOP=$1CA         ;ECS vertical line position for VSYNC stop
#VSSTRT=$1E0         ;ECS vertical line position for VSYNC start
#VTOTAL=$1C8         ;ECS highest numbered vertical line
```

·Amiga CIAs

The Amiga has two 8520 Complex Interface Adapter (CIA) which handle most of the Amiga I/O activities. Note that each register should be accessed as a byte and NOT a word. The following is an address map of both Amiga CIAs.

CIA A

ByteAddr	Register	b7	b6	b5	b4	b3	b2	b1	b0
$BFE001	pra	FIR1	FIR0	RDY	TK0	WPR0	CHNG	LED	OVL
$BFE101	prb	Parallel Port							
$BFE201	ddra	Direction for Port A (1=output)							
$BFE301	ddrb	Direction for Port B (1=output)							
$BFE401	talo	Timer A High Byte							
$BFE501	tahi	Timer A High Byte							
$BFE601	tblo	Timer B Low Byte							
$BFE701	tbhi	Timer B High Byte							
$BFE801	todlo	50/60 Hz Event Counter bits 7-0							
$BFE901	todmid	50/60 Hz Event Counter bits 15-8							
$BFEA01	todhi	50/60 Hz Event Counter bits 23-16							
$BFEB01		not used							
$BFEC01	sdr	Serial Data Register (connected to keyboard)							
$BFED01	icr	Interrupt Control Register							
$BFEE01	cra	Control Register A							
$BFEF01	crb	Control Register B							

CIA B

ByteAddr	Register	b7	b6	b5	b4	b3	b2	b1	b0
$BFD000	pra	DTR	RTS	CD	CTS	DSR	SEL	POUT	BUSY
$BFD100	prb	MTR	SEL3	SEL2	SEL1	SEL0	SIDE	DIR	STEP
$BFD200	ddra	Direction for Port A (1=output)							
$BFD300	ddrb	Direction for Port B (1=output)							
$BFD400	talo	Timer A High Byte							
$BFD500	tahi	Timer A High Byte							
$BFD600	tblo	Timer B Low Byte							
$BFD700	tbhi	Timer B High Byte							
$BFD800	todlo	Horizontal Sync Event Counter bits 7-0							
$BFD900	todmid	Horizontal Sync Event Counter bits 15-8							
$BFDA00	todhi	Horizontal Sync Event Counter bits 23-16							
$BFDB00		not used							
$BFDC00	sdr	Serial Data Register (connected to keyboard)							
$BFDD00	icr	Interrupt Control Register							
$BFDE00	cra	Control Register A							
$BFDF00	crb	Control Register B							

Apéndice 3: Lenguaje ensamblador de la familia 68000

Aquí se incluye una breve descripción de los elementos de programación en lenguaje ensamblador para la familia de CPU's 68000, que en esta arquitectura son los más comunes.

Nota: Los comentarios de esta compilación están en el idioma inglés, para conservar las ideas y significados originales y además poder distribuirlos en Internet si así se requiere.

·Registers

The 68000 has 6 internal registers, these may be thought of as high speed variables each capable of storing a long word (32 bits). The 8 data registers are used mainly for calculations while the 8 address registers are mostly used for pointing to locations in memory.

The registers are named D0-D7 and A0-A7. The 68000 also has several specialised registers, the *program counter (PC) and the status register (SR).* The program counter points to the current instruction that the microprocessor is executing, while the status register is a bunch of flags with various meanings.

·Addressing

The main job of the microprocessor is to read information from memory, perform a calculation and then write the result back to memory.

For the processor to access memory it has to generate a memory address for the location it wishes to access (read or write to). The following are the different ways the 68000 can generate addresses.

·Register Direct

```
MOVE d1,d0
```
The actual value in the register d1 is copied into d0

·Address Register Indirect

```
MOVE (a0),d0
```
a0 is a pointer to somewhere in memory. The value at at this location is copied into the register d0.

·*Address Register Indirect with Postincrement*

```
MOVE (a0)+,d0
```
The value at the location pointed to by a0 is copied into the register d0, then a0 is incremented so it points to the next memory location.

·Address Register Indirect with Predecrement

```
MOVE -(a0),d0
```
a0 is first decremented to point to the memory location before the one it currently points to, then the value at the new memory location is copied into d0.

·Address Register Indirect with Displacement

```
MOVE 16(a0),d0
```
The memory location located 16 bytes after that which is pointed to by address register a0 is copied to d0.

·Address Register Indirect with Index

```
MOVE 16(a0,d1),d0
```
The memory location is calculated by adding the contents of a0 with d1 plus 16.

·Absolute Address

```
MOVE $dff096,d0
```
The memory location $dff096 is used.

·Program Counter with Displacement

```
MOVE label(pc),d0
```
This is the same as absolute addressing but because the memory address is an offset from the program counter (no bigger than 32000 bytes) it is MUCH quicker.

·Program Counter with Index

```
MOVE label(pc,d1),d0
```
The address is calculated as the location of label plus the contents of data register d1.

·*Immediate Data*

```
MOVE #20,d0
```
The value 20 is moved to the data register.

·Program Flow

As mentioned previously the microprocessor has a special register known as the program counter that points to the next instruction to be executed. By changing the value in the program counter a 'goto' can be performed . The JMP instruction load the program counter with a new value, it supports most of the addressing modes.

A branch is a program counter relative form of the JMP instruction. Branches can also be performed on certain conditions such as BCC which will only cause the program flow to change if the Carry flag in the *status register is currently set.*

A 'gosub' can be performed using the JSR and BSR commands. The current value of the program counter is remembered on the stack before the jump or branch is performed. The RTS command is used to 'return' to the *original program location.*

·The Stack

The Amiga sets aside a certain amount of memory for each task known as a stack. The address register A7 is used to point to the stack and should never be used as a general purpose address register.

The 68000 uses predecrement addressing to push data onto the stack and postincrement addressing to pull information off the stack.

JSR is the same as MOVE.l pc,-(a7) and then JMP

RTS is the same as MOVE.l (a7)+,pc

The stack can be used to temporarily store internal registers. To save and restore all the 68000 registers the following code is often used:

```
ASubroutine:
MOVEM.l d0-d7/a0-a6,-(a7) ;push all registers on stack

;main sub. code here which can stuff up registers without worry.

MOVEM.l (a7)+,d0-d7/a0-a6 ;pull registers off stack
RTS ;return from subroutine
```

·Condition Flags

The status register is a special 68000 register that holds, besides other things all the condition codes. The following are a list of the condition flags:

Code	Name	Meaning
N	negative	reflects the most significant bit of the result of the last operation.
Z	zero	is set if the result is zero, cleared otherwise.
C	carry	is set when an add, subtract or compare operation generate a carry
X	extend	*is a mirror of the carry flag, however its not affected by data movement.*
V	overflow	is set when an aritmetic operation causes an overflow, a situation where the operand is not large enough to represent the result.

·Conditional Tests

Branches and Sets can be performed conditionally. The following is a list of the possible conditions that can be tested before a branch or set is performed.

cc	condition	coding	test
T	true	0000	1
F	false	0001	0
HI	*high*	0010	not C & not Z
LS	lowsam	0011	C | Z
CC	carry clr	0100	not C
CS	carry set	0101	C
NE	ot equal	0110	not Z
EQ	equal	0111	Z
VC	overflow clr	1000	not V
VS	overflow set	1001	V
PL	plus	1010	not N
MI	minus	1011	N
GE	greater equal	1100	N&V | notN&notV
LT	less than	1101	N&notV | notN&V
GT	greater than	1110	N&V&notZ | notN&notV&notC
LE	less or equal	1111	Z | N&notV | notN&V

·Operand Sizes

The 68000 can perform operations on bytes, words and long words. By adding a suffix .b .w or .l to the *opcode, the assembler knows which data size you wish to use, if no suffix is present the word size is* default. There is no speed increase using bytes instead of words as the 68000 classic is a 16 bit microprocessor and so no overhead is needed for 16 bit operations. However 32 bit long words do cause overhead with extra read and write cycles needed to perform operations on a bus that can only handle 16 bits at a time, anyway, newer microprocessors such as the 68020 and up can handle 32 bit wide instructions without worry.

· The 68000 Instruction Set

The following is a brief description of the 68000 instruction set.

Included with each are the addressing mode combinations available with each opcode. Their syntax are as follows:

Dn	data register
An	address register
Dy,Dx	data registers source & destination
Rx,Ry	register source & destination (data&address registers)
<ea>	effective address - a subset of addressing modes
#<data>	numeric constant

Special notes:

The address register operands ADDA, CMPA, MOVEA and SUBA are only word and long word data sizes. The last 'A' of the operand name is optional as it is with the immediate operands ADDI, CMPI, MOVEI, SUBI, ORI, EORI and ANDI.

The ADDQ and SUBQ are quick forms of their immediate cousins, The immediate data range is 1 to 8.

The MOVEQ instruction has a data range of -128 to 127 , the data is sign extended to 32 bits, and long is the only data size available.

The <ea> denotes an effective address, not all addressing modes are available with each effective address form of the instruction, as a rule program counter relative addressing is only available for the source operand and not the destination.

ABCD Add with extend using Binary Coded Decimal
ABCD Dy,Dx
ABCD -(Ay),-(Ax)
Data Size: byte

ADD Add binary
ADD <ea>,Dn
ADD Dn,<ea>
ADDA <ea>,An
ADDI #<data>,<ea>
ADDQ #<data>,<ea>
Data Size: byte, word & long

ADDX *Add with Extend*
ADDX Dy,Dx
ADDX -(Ay),-(Ax)
Data Size: byte, word & long

AND AND logical
AND <ea>,Dn
AND Dn,<ea>
ANDI #<data>,<ea>
Data Size: byte, word & long

ASL Arithmetic Shift Left
ASL Dx,Dy
ASL #<data>,Dy
ASL <ea>
Data Size: byte, word & long

ASR Arithmetic Shift Right
ASR Dx,Dy
ASR #<data>,Dy
ASR <ea>
Data Size: byte word & long

Bcc Branch Conditionally
Bcd <label>
Data Size: byte & word

BCHG Test a Bit & Change
BCHG Dn,<ea>
BCHG #<data>,<ea>
Data Size: byte & long

BCLR Test a Bit & Clear
BCLR Dn,<ea>
BCLR #<data>,<ea>
Data Size: byte & long

BRA Branch Always
BRA <label>
Data Size: byte & word

BSET Test a Bit & Set
BSET Dn,<ea>
BSET #<data>,<ea>
Data Size: byte & long

BTST Test a Bit
BTST Dn,<ea>
BTST #<data>,<ea>
Data Size: byte & long

CHK Check Register Against Bounds and TRAP
CHK <ea>,Dn
Data Size: word

CLR Clear an Operand
CLR <ea>
Data Size: byte, word & long

CMP Compare
CMP <ea>,Dn
CMPA <ea>,An
CMPI #<data>,<ea>
Data Size: byte, word & long

CMPM Compare Memory
CMPM (Ay)+,(Ax)+
Data Size: byte, word & long

DBcc Test Condition, Decrement, and Branch
DBcc Dn,<label>
Data Size: word

DIVS Signed Divide
DIVS <ea>,Dn Data
Data Size: word

DIVU Unsigned Divide
DIVU <ea>,Dn
Data Size: word

EOR Exclusive OR Logical
EOR Dn,<ea>
EORI #<data>,<ea>
Data Size: byte, word & long

EXG Exchange Registers
EXG Rx,Ry
Data Size: long

EXT Sign Extend
EXT Dn Data
Data Size: word & long

ILLEGAL Illegal Instruction
ILLEGAL
Data Size: none

JMP Jump
JMP <ea>
Data Size: long

JSR Jump to Subroutine
JSR <ea>
Data Size: long

LEA Load Effective Address
LEA <ea>,An
Data Size: long

LINK Link and Allocate
LINK An,#<displacement>
Data Size: word

LSL Logical Shift Left
LSL Dx,Dy
LSL #<data>,Dy
LSL <ea>
Data Size: byte, word & long

LSR Logical Shift Right
LSR Dx,Dy
LSR #<data>,Dy
LSR <ea>
Data Size: byte, word & long

MOVE Move Data from Source to Destination
MOVE <ea>,<ea>
MOVEA <ea>,An
MOVEQ #<data>,Dn
Data Size: byte, word & long

MOVEM Move Multiple Registers
MOVEM <register list>,<ea>
MOVEM <ea>,<register list>
Data Size: word & long

MOVEP Move Peripheral
MOVEP Dx,d(Ay)
MOVEP d(Ay),Dx
Data Size: word & long

MULS Signed Multiple
MULS <ea>,Dn
Data Size: word

MULU Unsigned Multiple
MULU <ea>,Dn
Data Size: word

NBCD Negate Decimal with Extend
NBCD <ea>
Data Size: byte

NEG Negate
NEG <ea>
Data Size: byte, word & long

NEGX Negate with Extend
NEGX <ea>
Data Size: byte, word & long

NOP No Operation
NOP
Data Size: none

NOT Logical Complement
NOT <ea>
Data Size: byte, word & long

OR Inclusive OR Logical
OR <ea>,Dn
OR Dn,<ea>
ORI #<data>,<ea>
Data Size: byte, word & long

PEA Push Effective Address
PEA <ea>
Data Size: long

RESET Reset External Device
RESET
Data Size: none

ROL Rotate Left (without Extend)
ROL Dx,Dy
ROL #<data>,Dn
ROL <ea>
Data Size: byte, word & long

ROR Rotate Right (without Extend)
ROR Dx,Dy
ROR #<data>,Dn
ROR <ea>
Data Size: byte, word & long

ROXL Rotate Left with Extend
ROXL Dx,Dy
ROXL #<data>,Dn
ROXL <ea>
Data Size: byte, word & long

ROXR Rotate Right with Extend
ROXR Dx,Dy
ROXR #<data>,Dn
ROXR <ea>
Data Size: byte, word & long

RTE Return from Exception
RTE Data
Data Size: None

RTR Return and Restore Condition Codes
RTR
Data Size: None

RTS Return from Subroutine
RTS
Data Size: None

SBCD Subtract Decimal with Extend
SBCD Dy,Dx
SBCD -(Ay),-(Ax)
Data Size: byte

Scc Set according to Condition
Scc <ea>
Data Size: byte

STOP Load Status Register and Stop
STOP #xxx
Data Size: None

SUB Subtract Binary
SUB <ea>,Dn
SUB Dn,<ea>
SUBA <ea>,An
SUBI #<data>,<ea>
SUBQ #<data>,<ea>
Data Size: byte, word & long

SUBX Subtract with Extend
SUBX Dy,Dx
SUBX -(Ay),-(Ax)
Data Size: byte, word & long

SWAP Swap Register Halves
SWAP Dn
Data Size: long

TAS Test & Set an Operand
TAS <ea>
Data Size: byte

TRAP Trap
TRAP #<vector>
Data Size: None

TRAPV Trap an Overflow
TRAPV
Data Size: None

TST Test an Operand
TST <ea>
Data Size: byte, word & long

UNLK Unlink
UNLK An Data
Data Size: None

Apéndice 4: Ejemplo del script utilizado en el sistema multimedia/video final

```
; ********************************
; * AT V1.0 Red UNAM Script      *
; * (C)1998 By Carlos Villarroel *
; ********************************
;=====
PART 1
;=====
;Transparent scroll and animation
;
LOAD_ANIM 1 work:render/anims/INTERTXT.ANIM
LOAD_BACKGROUND_MOV 1 work:render/pics/Internet_Photos1.iff+
,work:render/pics/Internet_Photos2.iff+
,work:render/pics/Internet_Photos3.iff+
,work:render/pics/Internet_Photos4.iff.0
LOAD_EXTRAFOREGROUND_MOV 1 work:render/pics/Telecom-BlueMAP.iff,0
;LOAD_VIRTUALGROUND_MOV 1 pics/NULL.mp,0
DISPLAY_ON
SPRITES_IN_FRONT
ENABLE_XMODE
GET_XBACKGROUND
ACTIVE_MOVS
SHOW_PAINTFILL 1 background
SHOW_GFADE_IN 64
WAIT 255
GET_XFOREOVERBACKGROUND
XFADE_SPEED 255
SHOW_XFADE 155
WAIT 155
SET_MAIN_FPS 30
PLAY_MAIN_ANIM 1 extraforeground
STOP_MAIN_ANIM
GET_XEXTRAFOREGROUND
XFADE_IN_SPEED 255
SHOW_XFADE 255
CONTINUE_MAIN_ANIM
WAIT_MAIN_FRAME 110
XFADE_OUT_SPEED 255
SHOW_XFADE 255
WAIT 255
SHOW_GFADE_OUT 255
WAIT 255
; End actions:
STOP_MAIN_ANIM
; the next to free move. mem:
LOAD_NULL_MOVS
ACTIVE_MOVS
; and the other mem:
; finally restore old display:
DISPLAY_OFF
FREE_ANIM 1
END
;
;=====
PART 2
;=====
;Final Credits Scroll
;
LOAD_BACKGROUND_MOV 2 Work:Render/Credits/MURAL_BG1.iff+
,Work:Render/Credits/MURAL_BG2.iff+
,Work:Render/Credits/MURAL_BG3.iff+
,Work:Render/Credits/MURAL_BG4.iff+
,Work:Render/Credits/MURAL_BG5.iff+
,Work:Render/Credits/MURAL_BG6.iff+
,Work:Render/Credits/MURAL_BG6.iff+
,Work:Render/Credits/MURAL_BG6.iff+
,Work:Render/Credits/MURAL_BG6.iff+
,Work:Render/Credits/MURAL_BG6.iff+
,Work:Render/Credits/MURAL_BG6.iff+
,Work:Render/Credits/MURAL_BG6.iff+
,Work:Render/Credits/MURAL_BG6.iff+
,Work:Render/Credits/MURAL_BG6.iff+
,Work:Render/Credits/MURAL_BG6.iff+
,Work:Render/Credits/MURAL_BG6.iff+
,Work:Render/Credits/MURAL_BG6.iff+
,0
LOAD_FOREGROUND_MOV 2 NULL.iff,Work:Render/Credits/MURAL_FG1.iff+
,Work:Render/Credits/MURAL_FG2.iff+
,Work:Render/Credits/MURAL_FG3.iff+
,Work:Render/Credits/MURAL_FG4.iff+
,Work:Render/Credits/MURAL_FG5.iff+
,Work:Render/Credits/MURAL_FG6.iff.NULL.iff+
,Work:Render/Credits/01MURALP37.iff,Work:Render/Credits/02MURALP41.iff+
,Work:Render/Credits/03MURALP36.iff,Work:Render/Credits/04MURALP35.iff+
,Work:Render/Credits/05MURALP33.iff,Work:Render/Credits/06MURALP31.iff+
,Work:Render/Credits/07MURALP29.iff,Work:Render/Credits/08MURALP21.iff+
,Work:Render/Credits/09MURALP22.iff,Work:Render/Credits/10MURALP24.iff+
,Work:Render/Credits/11MURALP25.iff,Work:Render/Credits/12MURALP19.iff+
,Work:Render/Credits/13MURALP18.iff,Work:Render/Credits/14MURALP06.iff+
,Work:Render/Credits/15MURALP13.iff,Work:Render/Credits/16MURALP05.iff+
,Work:Render/Credits/17MURALP06.iff,Work:Render/Credits/18MURALP07.iff+
,Work:Render/Credits/19MURALP03.iff,Work:Render/Credits/20MURALP01.iff+
,Work:Render/Credits/21MURALP10.iff,Work:Render/Credits/22MURALP01.iff+
,Work:Render/Credits/23MURALP08.iff+
,NULL.iff,Work:Render/Credits/MURALBP5_B.IFF+
,NULL.iff,Work:Render/Credits/MURALBP3_B.IFF+
,NULL.iff,Work:Render/Credits/MURALBP1_B.IFF,0
LOAD_VIRTUALGROUND_MOV 2 Work:Render/Credits/creditext1.spl+
,Work:Render/Credits/creditext2.spl,Work:Render/Credits/creditext1.spl+
,Work:Render/Credits/creditext2.spl,Work:Render/Credits/creditext1.spl+
,Work:Render/Credits/creditext3.spl,Work:Render/Credits/creditext3.spl+
,NULL.pal+
,Work:Render/Credits/creditext4.spl,Work:Render/Credits/creditext4.spl+
,NULL.pal+
,Work:Render/Credits/creditext5.spl,Work:Render/Credits/creditext5.spl+
,NULL.pal+
,0
SYSTEM_POINTER 268800 default
; --- Start external player (DeliTracker):
EXECUTE 0 run work:players/DeliTracker_II/Delitracker2 +
module Games:-CVVCMus/MDAT.Spydia_5-End +
playerpath work:players/Delitracker_II/deliplayers +
genispath work:players/Delitracker_II/deligenies cx_popup no
; --- And wait 1 second(s)
EXECUTE 0 wait 1
DISPLAY_ON
SPRITES_IN_FRONT
DISABLE_XMODE
ACTIVE_MOVS
AUTO_FG_PAL_FADE
GET_BACKGROUND
GET_FOREGROUND
GET_SPRITES
;SHOW_PAINTFILL 1 background
;SHOW_GFADE_IN 255
SHOW
WAIT 1230
SHOW_GFADE_OUT 255
WAIT 255
; End actions:
LOAD_NULL_MOVS
ACTIVE_MOVS
; and the other mem:
; finally restore old display:
DISPLAY_OFF
; --- And quit external player (Delitracker):
EXECUTE 0 status>ram:dt.tmp COMMAND=work:players/DeliTracker_II/Delitracker2
EXECUTE 0 break <ram:dt.tmp >NIL: ?
; ---
END
;
;=====
PART 4
;=====
;Initial Introduction
;
LOAD_ANIM 1 Work:render/anims/WORLD.ANIM
LOAD_MUS 1 Games:Samples/CVVC_MODS/MOD.ShadowFighter_Aussie
LOAD_BACKGROUND_MOV 4 NULL.iff,NULL.iff,0
LOAD_FOREGROUND_MOV 4 NULL.iff,work:render/pics/RedUNAM_BigTx1.iff+
,work:render/pics/RedUNAM_BigTx2.iff,work:render/pics/RedUNAM_BigTx3.iff+
,work:render/pics/RedUNAM_BigTx4.iff,work:render/pics/RedUNAM_BigTx5.iff+
,work:render/pics/RedUNAM_BigTx6.iff,work:render/pics/RedUNAM_BigTx7.iff.0
;LOAD_VIRTUALGROUND_MOV 4 work:render/pics/IntroText.spl+
;,work:render/pics/IntroText.mp2+
;,work:render/pics/IntroText.pal.0
;SPRITES_IN_FRONT
DISABLE_XMODE
SET_MAIN_FPS 30
GET_FOREGROUND
GET_SPRITES
SOUND_OFF
MUS_VOL 64
PLAY_MUS 1
DISPLAY_ON
ACTIVE_MOVS
PLAY_MAIN_ANIM 1 background
GET_BACKGROUND
SHOW
SHOW_PAINTFILL 1 foreground
GET_BLANK
WAIT_MAIN_FRAME 147
;Test: WAIT_MAIN_FRAME 160
MUS_VOL_FADE 30 10
FADE_SPEED 255
SHOW_FADE 255
WAIT 255
; End actions:
STOP_MAIN_ANIM
LOAD_NULL_MOVS
ACTIVE_MOVS
; The command SOUND_ON let atsoundplayer work:
SOUND_ON
; and the other mem:
; finally restore old display:
SYSTEM_POINTER 268800 default
DISPLAY_OFF
FREE_ANIM 1
EXECUTE 0 run atsoundplayer Work2:SCALA_STUFF/Sounds/Jingle001
END
;
;=====
PART 6
;=====
;RIT/EKD-UNAM presentation
;
LOAD_BACKGROUND_MOV 6 work:render/pics/RUNAM_PARALLAX2a.iff+
,work:render/pics/RUNAM_PARALLAX2b.iff+
,work:render/pics/RUNAM_PARALLAX2c.iff+
,work:render/pics/RUNAM_PARALLAX2a.iff.0
LOAD_FOREGROUND_MOV 6 NULL.iff+
,work:render/pics/RUNAM_TEXT.iff+
,work:render/pics/RUNAM_PARALLAX1a.iff+
,work:render/pics/RUNAM_PARALLAX1b.iff+
,work:render/pics/RUNAM_TEXT.iff+
,work:render/pics/RUNAM_PARALLAX1a.iff+
,work:render/pics/RUNAM_PARALLAX1b.iff+
,work:render/pics/RUNAM_TEXT.iff+
,work:render/pics/RUNAM_PARALLAX1a.iff+
```

```
,work:render/pics/RUNAM_PARALLAX1b.iff+
,work:render/pics/RUNAM_TEXT.iff+
,NULL.iff+
,work:render/pics/RUNAM_PARALLAX1b.iff+
,NULL.iff+
,work:render/pics/RUNAM_TEXT.iff+
,work:render/pics/RUNAM_PARALLAX1a.iff+
,work:render/pics/RUNAM_PARALLAX1b.iff+
,work:render/pics/RUNAM_TEXT.iff+
,work:render/pics/RUNAM_PARALLAX1a.iff+
,work:render/pics/RUNAM_PARALLAX1b.iff+
,work:render/pics/RUNAM_TEXT.iff+
,work:render/pics/RUNAM_PARALLAX1a.iff+
,work:render/pics/RUNAM_PARALLAX1b.iff+
,work:render/pics/RUNAM_TEXT.iff+
,work:render/pics/RUNAM_PARALLAX1a.iff+
,work:render/pics/RUNAM_PARALLAX1b.iff+
,work:render/pics/RUNAM_TEXT.iff+
,0
LOAD_VIRTUALGROUND_MOV 6 work:render/pics/RUNAM_PARALLAX3a1.sp1+
,work:render/pics/RUNAM_PARALLAX3a1.sp1+
,work:render/pics/RUNAM_PARALLAX3a1.sp2+
,work:render/pics/RUNAM_PARALLAX3a1.sp3+
,work:render/pics/RUNAM_PARALLAX3a1.pal,0
DISPLAY_ON
SPRITES_IN_BACK
DISABLE_XMODE
ACTIVE_MOVS
GET_FOREGROUND
FADE_SPEED 255
SHOW_FADE 255
WAIT 255
GET_BACKGROUND
FADE_SPEED 255
SHOW_FADE 255
WAIT 255
GET_SPRITES
FADE_SPEED 255
SHOW_FADE 255
WAIT 255
WAIT 400
SHOW_GFADE_OUT 255
WAIT 255
; End actions:
LOAD_NULL_MOVS
ACTIVE_MOVS
; and the other mem:
; finally restore old display:
SYSTEM_POINTER 268800 default
DISPLAY_OFF
END
;
;=====
PART 8
;=====
;NOC presentation
;
LOAD_BACKGROUND_MOV 8 work:render/pics/DiagBackgr1.iff+
,work:render/pics/DiagBackgr3.iff+
,work:render/pics/DiagBackgr2.iff+
,work:render/pics/DiagBackgr1.iff,0
LOAD_FOREGROUND_MOV 3 work:render/pics/NettedBackgr.iff+
,work:render/pics/NettedBackgr.iff+
,work:render/pics/NOC.iff,0
LOAD_VIRTUALGROUND_MOV 8 NULL.sp,NULL.sp,NULL.pal+
,NULL.sp,NULL.pal+
,work:render/pics/NOC_Sign6.sp,work:render/pics/NOC_Sign6.pal+
,work:render/pics/NOC_Sign5.sp,NULL.pal+
,work:render/pics/NOC_Sign4.sp,NULL.pal+
,work:render/pics/NOC_Sign3.sp,NULL.pal+
,work:render/pics/NOC_Sign2.sp,NULL.pal+
,work:render/pics/NOC_Sign1.sp,NULL.pal+
,0
DISPLAY_ON
SPRITES_IN_FRONT
ENABLE_XMODE
ACTIVE_MOVS
GET_XBACKGROUND
SHOW_GFADE_IN 64
WAIT 64
GET_XFOREOVERBACKGROUND
XFADE_SPEED 180
SHOW_XFADE 180
WAIT 180
GET_BACKGROUND
GET_FOREGROUND
GET_SPRITES
DISABLE_XMODE
WAIT 600
SHOW_GFADE_OUT 255
WAIT 255
; End actions:
LOAD_NULL_MOVS
ACTIVE_MOVS
; and the other mem:
; finally restore old display:
DISPLAY_OFF
END
;
;=====
PART 9
;=====
;TAC presentation
;
LOAD_BACKGROUND_MOV 8 work:render/pics/DiagBackgr1.iff+
,work:render/pics/DiagBackgr3.iff+
,work:render/pics/DiagBackgr2.iff+
,work:render/pics/DiagBackgr1.iff,0
LOAD_FOREGROUND_MOV 3 work:render/pics/NettedBackgr.iff+
,work:render/pics/NettedBackgr.iff+
,work:render/pics/TAC.iff,0
LOAD_VIRTUALGROUND_MOV 8 NULL.sp,NULL.sp,NULL.pal+
,NULL.sp,NULL.pal+
,work:render/pics/TAC_Sign6.sp,work:render/pics/TAC_Sign6.pal+
,work:render/pics/TAC_Sign5.sp,NULL.pal+
,work:render/pics/TAC_Sign4.sp,NULL.pal+
,work:render/pics/TAC_Sign3.sp,NULL.pal+
,work:render/pics/TAC_Sign2.sp,NULL.pal+
,work:render/pics/TAC_Sign1.sp,NULL.pal+
,0
DISPLAY_ON
SPRITES_IN_FRONT
ENABLE_XMODE
ACTIVE_MOVS
GET_XBACKGROUND
SHOW_GFADE_IN 64
;SHOW_PAINTFILL 1 background
WAIT 64
GET_XFOREOVERBACKGROUND
XFADE_SPEED 180
SHOW_XFADE 180
WAIT 180
GET_BACKGROUND
GET_FOREGROUND
GET_SPRITES
DISABLE_XMODE
WAIT 600
SHOW_GFADE_OUT 255
WAIT 255
; End actions:
LOAD_NULL_MOVS
ACTIVE_MOVS
; and the other mem:
; finally restore old display:
DISPLAY_OFF
END
;
;======
PART 10
;======
;NIC presentation
;
LOAD_BACKGROUND_MOV 8 work:render/pics/DiagBackgr1.iff+
,work:render/pics/DiagBackgr3.iff+
,work:render/pics/DiagBackgr2.iff+
,work:render/pics/DiagBackgr1.iff,0
LOAD_FOREGROUND_MOV 3 work:render/pics/NettedBackgr.iff+
,work:render/pics/NettedBackgr.iff+
,work:render/pics/NIC.iff,0
LOAD_VIRTUALGROUND_MOV 8 NULL.sp,NULL.sp,NULL.pal+
,NULL.sp,NULL.pal+
,work:render/pics/NIC_Sign6.sp,work:render/pics/NIC_Sign6.pal+
,work:render/pics/NIC_Sign5.sp,NULL.pal+
,work:render/pics/NIC_Sign4.sp,NULL.pal+
,work:render/pics/NIC_Sign3.sp,NULL.pal+
,work:render/pics/NIC_Sign2.sp,NULL.pal+
,work:render/pics/NIC_Sign1.sp,NULL.pal+
,0
DISPLAY_ON
SPRITES_IN_FRONT
ENABLE_XMODE
ACTIVE_MOVS
GET_XBACKGROUND
SHOW_GFADE_IN 64
;SHOW_PAINTFILL 1 background
WAIT 64
GET_XFOREOVERBACKGROUND
XFADE_SPEED 180
SHOW_XFADE 180
WAIT 180
GET_BACKGROUND
GET_FOREGROUND
GET_SPRITES
DISABLE_XMODE
WAIT 600
SHOW_GFADE_OUT 255
WAIT 255
; End actions:
LOAD_NULL_MOVS
ACTIVE_MOVS
; and the other mem:
; finally restore old display:
DISPLAY_OFF
END
;
;======
PART 11
;======
;Admon. de Servidores presentation
;
LOAD_BACKGROUND_MOV 8 work:render/pics/DiagBackgr1.iff+
,work:render/pics/DiagBackgr3.iff+
,work:render/pics/DiagBackgr2.iff+
,work:render/pics/DiagBackgr1.iff,0
LOAD_FOREGROUND_MOV 8 work:render/pics/NettedBackgr.iff+
,work:render/pics/NettedBackgr.iff+
,work:render/pics/AdminServidores.iff,0
DISPLAY_ON
SPRITES_IN_FRONT
ENABLE_XMODE
ACTIVE_MOVS
GET_XBACKGROUND
SHOW_GFADE_IN 64
;SHOW_PAINTFILL 1 background
WAIT 64
GET_XFOREOVERBACKGROUND
```

```
XFADE_SPEED 180
SHOW_XFADE 180
WAIT 180
GET_BACKGROUND
GET_FOREGROUND
GET_SPRITES
DISABLE_XMODE
WAIT 600
SHOW_GFADE_OUT 255
WAIT 255
; End actions:
LOAD_NULL_MOVS
ACTIVE_MOVS
; and the other mem:
; finally restore old display:
DISPLAY_OFF
END
;
;******
PART 12
;******
;Atancion a usuarios presentation
;
LOAD_BACKGROUND_MOV 5 work:render/pics/DiagBackgr1.iff+
,work:render/pics/DiagBackgr3.iff+
,work:render/pics/DiagBackgr2.iff+
,work:render/pics/DiagBackgr1.iff,0
LOAD_FOREGROUND_MOV 8 work:render/pics/NettedBackgr.iff+
,work:render/pics/NettedBackgr.iff+
,work:render/pics/AtencionUsuarios.iff,0
DISPLAY_ON
SPRITES_IN_FRONT
ENABLE_XMODE
ACTIVE_MOVS
GET_XBACKGROUND
SHOW_GFADE_IN 64
;SHOW_PAINTFILL 1 background
WAIT 64
GET_XFOREOVERBACKGROUND
XFADE_SPEED 180
SHOW_XFADE 180
WAIT 180
GET_BACKGROUND
GET_FOREGROUND
GET_SPRITES
DISABLE_XMODE
WAIT 600
SHOW_GFADE_OUT 255
WAIT 255
; End actions:
LOAD_NULL_MOVS
ACTIVE_MOVS
; and the other mem:
; finally restore old display:
DISPLAY_OFF
END
;
;******
PART 13
;******
;Conectividad presentation
;
LOAD_BACKGROUND_MOV 8 work:render/pics/DiagBackgr1.iff+
,work:render/pics/DiagBackgr3.iff+
,work:render/pics/DiagBackgr2.iff+
,work:render/pics/DiagBackgr1.iff,0
LOAD_FOREGROUND_MOV 8 work:render/pics/NettedBackgr.iff+
,work:render/pics/NettedBackgr.iff+
,work:render/pics/Conactividad.iff,0
DISPLAY_ON
SPRITES_IN_FRONT
ENABLE_XMODE
ACTIVE_MOVS
GET_XBACKGROUND
SHOW_GFADE_IN 64
;SHOW_PAINTFILL 1 background
WAIT 64
GET_XFOREOVERBACKGROUND
XFADE_SPEED 180
SHOW_XFADE 180
WAIT 180
GET_BACKGROUND
GET_FOREGROUND
GET_SPRITES
DISABLE_XMODE
WAIT 600
SHOW_GFADE_OUT 255
WAIT 255
; End actions:
LOAD_NULL_MOVS
ACTIVE_MOVS
; and the other mem:
; finally restore old display:
DISPLAY_OFF
END
;
;******
PART 14
;******
;DTD presentation
;
LOAD_BACKGROUND_MOV 14 work:render/pics/DiagBackgr1.iff+
,work:render/pics/DiagBackgr3.iff+
,work:render/pics/DiagBackgr2.iff+
,work:render/pics/DiagBackgr1.iff,0
LOAD_FOREGROUND_MOV 14 NULL.iff+
,work:render/pics/DTD.iff+
,work:render/pics/DTD Organigrama.iff,0
LOAD_VIRTUALGROUND_MOV 14 NULL.sp+
,work:render/pics/DTD_Sign3.sp,work:render/pics/DTD_Sign3.pal+
,work:render/pics/DTD_Sign2.sp,NULL.pal+
,work:render/pics/DTD_Sign1.sp,NULL.pal+
,0
DISPLAY_ON
SPRITES_IN_FRONT
DISABLE_XMODE
GET_BACKGROUND
GET_FOREGROUND
; Next 2 lines. Odd: ACTIVE-GETSP GETSP-ACTIVE
ACTIVE_MOVS
GET_SPRITES
SHOW_GFADE_IN 64
SHOW_PAINTFILL 1 background
WAIT 1200
SHOW_GFADE_OUT 255
WAIT 255
; End actions:
LOAD_NULL_MOVS
ACTIVE_MOVS
; and the other mem:
; finally restore old display:
DISPLAY_OFF
END
;
;******
PART 15
;******
;WWW anim
;
LOAD_ANIM 1 work:render/anims/www.anim
LOAD_BACKGROUND_MOV 15 NULL.iff,NULL.iff,0
LOAD_FOREGROUND_MOV 15 work:render/pics/WWW_Photos4.iff+
,work:render/pics/WWW_Photos1.iff+
,work:render/pics/WWW_Photos2.iff+
,work:render/pics/WWW_Photos3.iff,0
DISPLAY_ON
SPRITES_IN_FRONT
ENABLE_XMODE
ACTIVE_MOVS
SET MAIN_FPS 30
PLAY_MAIN_ANIM 1 background
STOP_MAIN_ANIM
GET_XBACKGROUND
CONTINUE MAIN_ANIM
SHOW_PAINTFILL 1 background
SHOW_GFADE_IN 200
WAIT 200
;
GET_XFOREOVERBACKGROUND
XFADE_SPEED 255
SHOW_XFADE 155
WAIT 750;
SHOW_GFADE_OUT 255
WAIT 255
; End actions:
STOP_MAIN_ANIM
; the next to free movs. mem:
LOAD_NULL_MOVS
ACTIVE_MOVS
; and the other mem:
; finally restore old display:
DISPLAY_OFF
FREE_ANIM 1
END
;
;******
PART 16
;******
;E-Mail anim
;
LOAD_BACKGROUND_MOV 16 work:render/pics/EMail2_BLUE.iff+
,work:render/pics/EMail2_BLUE.iff+
,work:render/pics/EMail2_BLUE.iff+
,work:render/pics/EMail2_BLUE.iff,0
LOAD_FOREGROUND_MOV 16 work:render/pics/EMail_address1.iff+
,work:render/pics/EMail_address2.iff+
,work:render/pics/EMail_address3.iff,0
DISPLAY_ON
SPRITES_IN_FRONT
ENABLE_XMODE
ACTIVE_MOVS
GET_XFOREGROUND
SHOW
GET_XFOREOVERBACKGROUND
XFADE_SPEED 255
SHOW_XFADE 255
WAIT 255
WAIT 900
SHOW_GFADE_OUT 255
WAIT 255
; End actions:
LOAD_NULL_MOVS
ACTIVE_MOVS
; and the other mem:
; finally restore old display:
DISPLAY_OFF
END
;

;******
PART 17
;******
;A500 animation
;
LOAD_ANIM 1 work:render/Anims/A500.ANIM
LOAD_BACKGROUND_MOV 17 work:render/pics/A500Backgr1.iff+
```

```
;work:render/pics/A500Backgr2.iff+
;work:render/pics/A500Backgr3.iff,0
LOAD_FOREGROUND_MOV 17 NULL.iff,NULL.iff,0
LOAD_VIRTUALGROUND_MOV 17 work:render/pics/Earth1c4.spl+
;work:render/pics/Earth1c3.spl+
;work:render/pics/Earth1c4.sp2+
;work:render/pics/Earth1c1.sp2+
;work:render/pics/Earth1c3.pal+
;work:render/pics/Earth1c1.spl+
;work:render/pics/Earth1c2.sp2+
;work:render/pics/Earth1c2.pal+
;work:render/pics/Earth1c1.spl+
;work:render/pics/Earth1c1.sp2+
;work:render/pics/Earth1c1.pal,0
SPRITES_IN_MID
DISABLE_KMODE
SET MAIN_FPS 1
GET BACKGROUND
GET_SPRITES
DISPLAY_ON
ACTIVE MOVS
MAIN ANIM7
PLAY MAIN ANIM 1 foreground
GET FOREGROUND
SHOW_GFADE_IN 128
WAIT 255
WAIT 335
SHOW GFADE OUT 128
WAIT 108
STOP_MAIN ANIM
WAIT 20
; End actions:
LOAD NULL MOVS
ACTIVE MOVS
; and the other mem:
; finally restore old display:
DISPLAY OFF
FREE ANIM 1
END
;
;******
PART 19
;******
;IRC animation
;
LOAD_ANIM 1 work:render/anims/IRC.ANIM
LOAD_BACKGROUND MOV 19 work:render/pics/IRC.iff,0
LOAD_FOREGROUND MOV 19 NULL.iff,0
ENABLE_KMODE
SET MAIN_FPS 14
GET KBACKGROUND
DISPLAY ON
ACTIVE MOVS
SHOW GFADE IN 64
PLAY MAIN ANIM 1 foreground
STOP MAIN ANIM
WAIT 64
GET XFOREGROUND
XFADE SPEED 128
SHOW_XFADE 128
WAIT 64
CONTINUE_MAIN_ANIM
WAIT_MAIN_FRAME 65
STOP_MAIN_ANIM
DISPLAY_SYSTEM 128
CONTINUE_MAIN_ANIM
WAIT_MAIN_FRAME 162
STOP_MAIN_ANIM
; End actions:
LOAD_NULL_MOVS
ACTIVE_MOVS
; and the other mem:
; finally restore old display:
DISPLAY_OFF
FREE_ANIM 1
END
;
;******
PART 20
;******
;Main mouu pointer on left/down corner
SYSTEM_POINTER 268800 default
END
;
;******
PART 21
;******
;Trumpet/TCP connection icon
SYSTEM_POINTER 12800 work:developer/Pointers/TCP_ON.Ptr
END
;
;******
PART 22
;******
;Telnet icon
SYSTEM_POINTER 12800 work:developer/Pointers/Telnet.Ptr
END
;
;******
PART 23
;******
;EMail icon
SYSTEM_POINTER 12800 work:developer/Pointers/EMail.Ptr
END
;
;******
PART 24
;******
;FTP icon
SYSTEM_POINTER 12800 work:developer/Pointers/FTP.Ptr
END
;
;******
PART 25
;******
;WWW icon
SYSTEM_POINTER 12800 work:developer/Pointers/WWW.Ptr
END
;
;******
PART 26
;******
;Trumpet/TCP disconnection icon
SYSTEM_POINTER 12800 work:developer/Pointers/TCP_OFF.Ptr
END
;
```