De Re Atari

UNA GUIA

A LA

PROGRAMACION ATARI

De Re Atari

una guìa para

la programación eficaz

con el Sistema

de Computación Personal

Atari

C O N T E N I D O

VOLUMEN I

VOLUMEN II

VOLUMEN III

PREFACIO

Este manual trata del Sistema de Computación Personal ATARI. Cubre tanto los modelos 400 y 800 como los XL. Los modelos 400 y 800 son electricamente idénticos, difiriendo fundamentalmente en características mecánicas como el teclado, las ranuras de cartridge y algunas conecciones externas. Las diferencias entre los modelos 400 , 800 y los XL son mayores, no sólo existen algunas diferencias mecánicas y eléctricas, sino que además diferencias de software y hardware, especialmente en lo que se refiere al Sistema Operativo, las puertas de controladores y teclas de funciones respectivamente. En los casos en que corresponda, estas diferencias se destacarán a lo largo del libro. El propósito es explicar en detalle como usar todas las posibilidades del Sistema de Computación Personal Atari. Se trata de un producto complejo y poderoso y por lo tanto las explicaciones serán necesariamente bastante largas. Además requieren por parte del lector algún conocimiento previo.

Este libro no se dirige al programador novicio; el lector deberá estar familiarizado con el Manual de Referencia Basic ATARI. También se requiere una cierta familiaridad con lenguaje Assembler.
Un glosario, que se encuentra al final, define y explica algunos de los términos menos usuales; sin embargo, este glosario no incluye los términos que todo programador serio en el campo de los computadores personales debiera conocer desde ya. El libro fue escrito como un manual de entrenamiento para programadores profesionales, que desean utilizar el Sistema de Computación Personal ATARI; no sustituye de ninguna forma los manuales de referencia técnica, los cuales son muy útiles para programadores que ya conocen y comprenden el sistema.

Este libro pretende ser de carácter tutorial, explica ideas y posibilidades más que define registros y códigos de control. Fue escrito originalmente por el entonces grupo de soporte de desarrollo de software. Chris Crawford escribió los Capìtulos 1 al 6 y los Apéndices I y II. Lane Winner escribió el Capìtulo 7 y el Apéndice IV, con la ayuda de Jim Cox. Amy Chen escribió el Apéndice III. Mike Ekberg, con la asistencia de John Eckstrom, escribió los Capìtulos 8 y 9. Kathleen Pitta escribió el Apéndice X. Bob Fraser escribió el Capìtulo 10. Gus Makreas preparó el glosario y el Contenido. El resultado final podrá tener deficiencias, pero ellos están orgullos de él. Con posterioridad, y durante la traducción se incorporaron las referencias especìficas relacionadas con los modelos XL.

CAPITULO 1
VISTA GENERAL DEL SISTEMA

El Sistema de Computación Personal ATARI es un computador personal de la segunda generación. Primero y ante todo es un computador para el usuario general. El énfasis del diseño se ha puesto en hacer sentirse a gusto al consumidor con el computador. Esta orientación hacia el consumidor se manifiesta en muchas formas. Primero, la máquina es a prueba de idiotas; al usuario se le protege de errores por medio de cosas como conectores polarizados, que no pueden enchufarse mal, protección de inserción de cartridges y por ejemplo la ubicación aislada de la tecla de reposición del sistema SYSTEM RESET. En segundo lugar, la máquina tiene un gran potencial gráfico; la gente responde a las imágenes mucho mejor que al texto. Tercero, la máquina tiene capacidades de sonido; nuevamente la gente responde mejor a este tipo de llamado que a mensajes indirectos de texto. Por último, el computador tiene bastones de control y paletas para ingresos táctiles más directos de lo que es posible con un teclado; los mismos conectores para bastones y paletas permiten la conección de otros periféricos de entrada/salida de caracterìsticas particulares para aplicaciones especìficas. El punto aquì es, que no se trata de que el computador tenga una serie de ventajas aisladas, sino que todas estas ventajas formen parte de una filosofìa de diseño consistente, orientada directamente hacia el consumidor. El diseñador que no aprecie asì este hecho fundamental, se encontrará trabajando contra las caracterìsticas del sistema.

La realización interna de estos computadores es muy diferente a la de otros sistemas. Naturalmente posee un microprocesador (un 6502), RAM, ROM y una PIA (o dos en el caso de los modelos 400/800). Sin embargo, además tiene tres circuitos integrados de gran escala (LSI) de propósitos esbeciales, conocidos como ANTIC, POKEY, y GTIA (Los primeros computadores ATARI tenìan un circuito CTIA , menos poderoso que el GTIA). Estos circuitos fueron diseñados por ingenieros de Atari, principalmente para descargar al 6502 de parte de su labor doméstica, liberándolo asì para que se concentrara en las computaciones propiamente tales. Al hacerlo incluyeron en estos circuitos integrados una gran potencia. Cada uno de ellos es casi tan grande (en términos de área de silicio) como el 6502, de modo que los tres en conjunto poseen un poder enorme. Dominar el Atari es fundamentalmente cosa de dominar estos tres circuitos integrados.

ANTIC es un microprocesador dedicado al despliegue de

imágenes de televisión. Es de hecho un microprocesador; tiene un juego de instrucciones, un programa (llamado la lista de despliegue), y datos. La lista y los datos de despliegue son escritos en RAM por el 6502. ANTIC encuentra la información de la RAM, utilizando acceso de memoria directo (DMA). Procesa las instrucciones de nivel superior contenidas en la lista de despliegue y traduce estas instrucciones a una corriente de instrucciones simples en tiempo real para la GTIA.

GTIA es un circuito de interfaz de televisión. ANTIC controla directamente la mayoría de las operaciones de la GTIA, pero a su vez se puede programar al 6502 para intervenir y controlar algunas o todas las funciones de GTIA. La GTIA convierte los comandos digitales de ANTIC (o 6502) en señales que van hacia el televisor. GTIA agrega también algunos factores propios, como valores de color, gráficos de jugador-proyectil y detección de colisiónes.

POKEY es un circuito digital de entrada/salida. Está a cargo de tareas tan dispares como el bus serial de entrada/salida, la generación de audio, exploración del teclado y generación de números aleatorios. También digitaliza los ingresos de paleta resistivos y controla los requerimientos de interrupcioón enmascarables (IRQ) de los periféricos.

Estos cuatro circuitos integrados de gran escala operan en forma simultánea. Una separación cuidadosa de sus funciones durante la fase de diseño ha minimizado los conflictos que puedan existir entre ellos. El único conflicto de tipo circuital que existe entre dos de estos integrados en el sistema ocurre cuando ANTIC requiere usar los buses de data y de direcciones para obtener su información de despliegue. Para hacerlo detiene la operación del 6502 y toma control de los buses.

Como en todos los sistemas 6502, la entrada y salida es a través de direcciones de memoria. La Figura 1.1 presenta un mapa de memoria grueso para el computador. La Figura 1.2 muestra el agrupamiento de los circuitos.

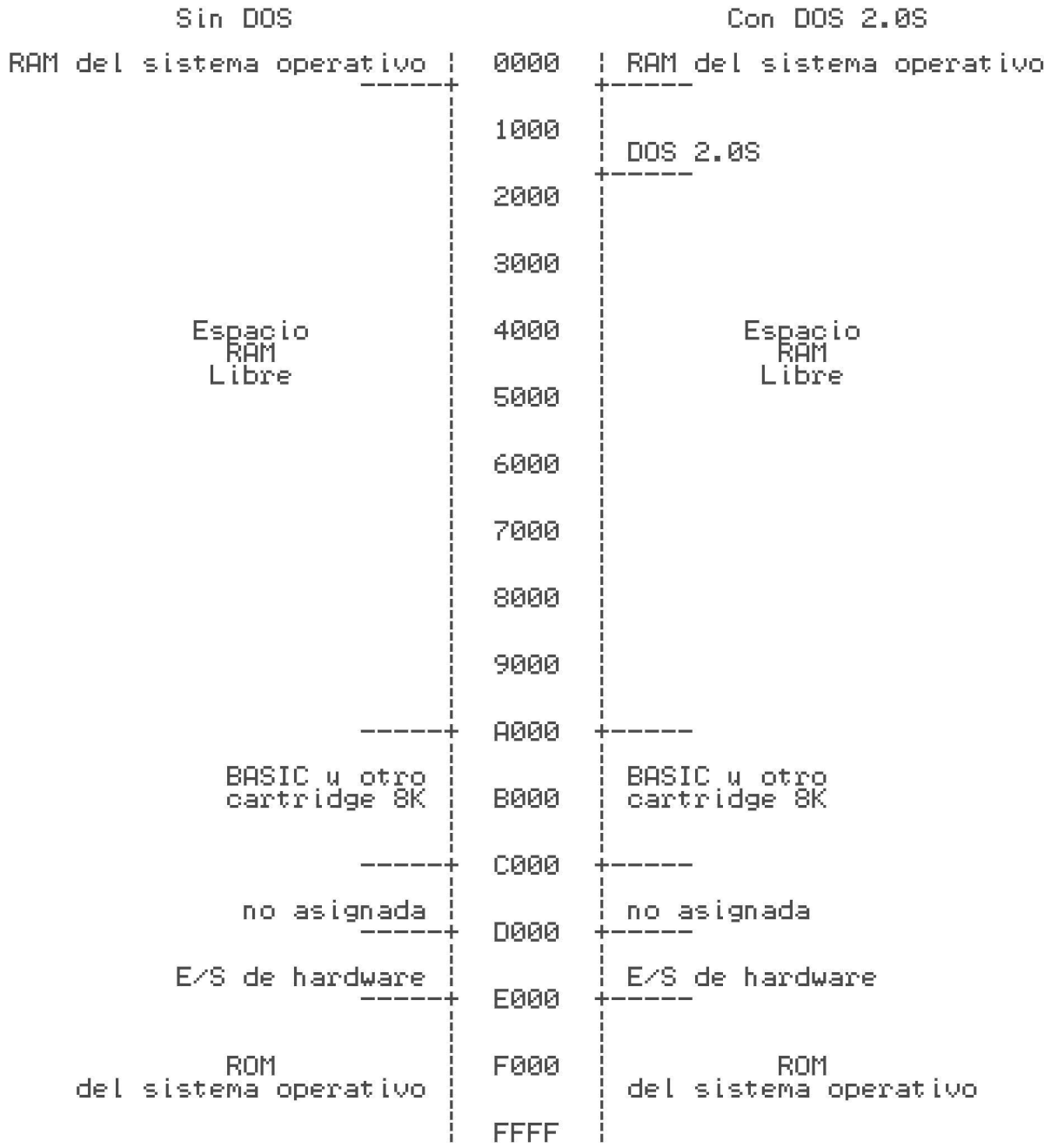


Figura 1.1
Disposicion de la memoria de Atari

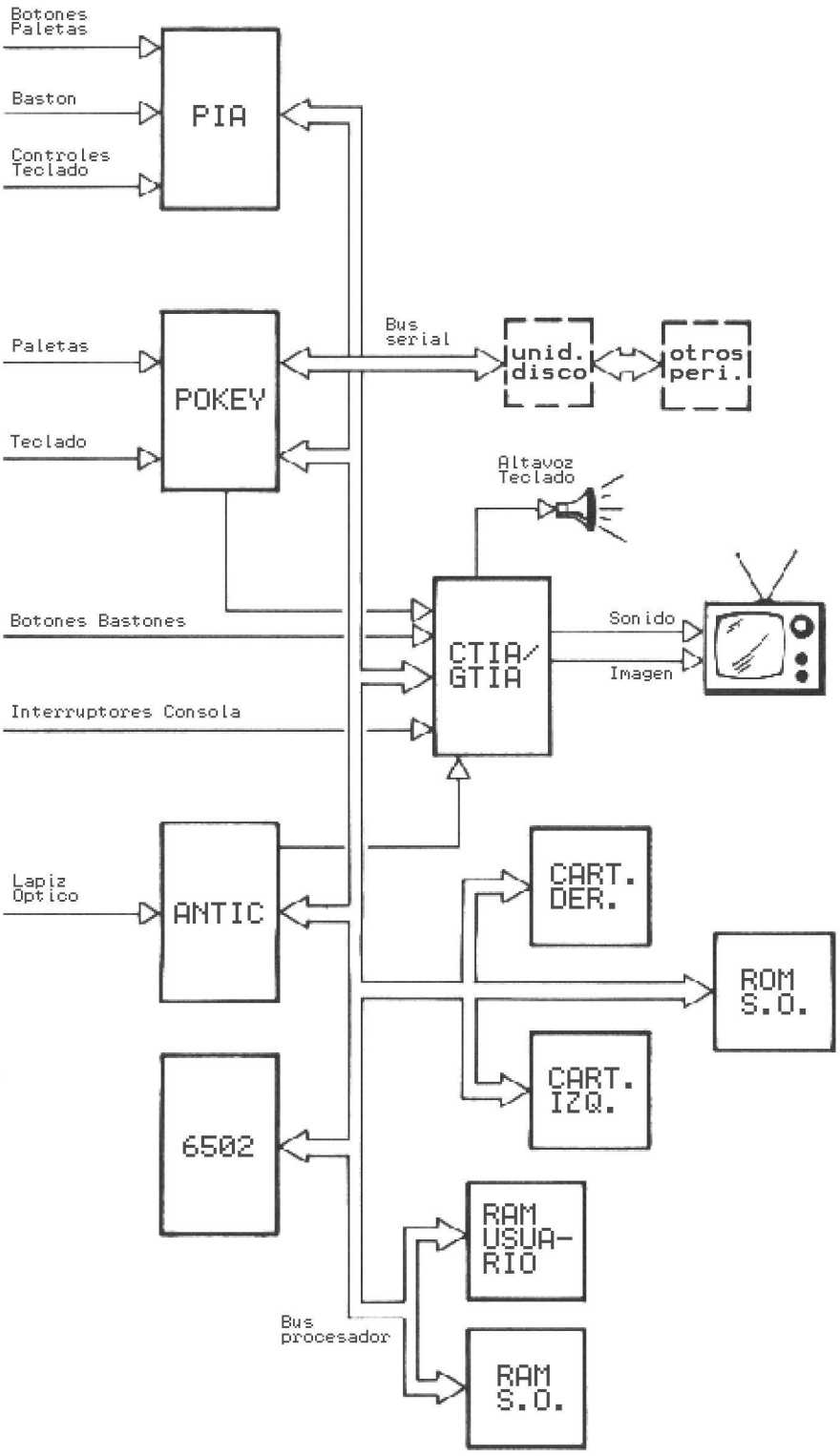


Figura 1.2
Disposicion del hardware de Atari

CAPITULO 2
ANTIC Y LA LISTA DE DESPLIEGUE

DESPLIEGUES DE TELEVISION

Para comprender las capacidades gráficas del Sistema de Computación Personal ATARI, primero debe entenderse los rudimientos de como funciona un receptor de televisión. Los receptores de televisión emplean lo que se llama un sistema de despliegue por barrido. En la parte posterior del tubo de televisión se genera un haz de electrones que es disparado hacia la pantalla del tubo. A lo largo de su camino, pasa a través de dos juegos de bobinas, el horizontal y el vertical, los que, si están energizados, pueden desviar el haz. Así puede conseguirse que el haz toque la pantalla en cualquier punto. La circuitería electrónica al interior del televisor hace que el haz barra a través de la pantalla en una forma regular. También la intensidad del haz puede controlarse. Si el haz es más intenso, el punto sobre la pantalla brillará en forma más intensa; al ser menos intenso el haz, el punto brillará en forma más débil o simplemente no se verá.

Durante el funcionamiento, el haz parte en el rincón superior izquierdo de la pantalla y la cruza horizontalmente. A medida que lo hace, su intensidad cambia, pintando así una parte de la imagen sobre la pantalla. Al llegar a su extremo derecho, se apaga y vuelve hacia el costado izquierdo. Al mismo tiempo baja una pequeña cantidad, enseguida vuelve a conectarse y nuevamente barre a través de la pantalla. Este proceso se repite hasta obtener un total de 262 barridos (De hecho hay 525 pasadas a través de la pantalla en un sistema alternativo conocido como entrelazado. Ignoraremos el entrelazado y actuaremos como si la televisión tuviera solamente 262 líneas). Estas 262 líneas llenan la pantalla de arriba hasta abajo. En la parte baja de la pantalla, una vez trazada la línea número 262, el haz de electrones se desconecta y vuelve al rincón superior izquierdo de la pantalla. En ese momento reinicia el ciclo. Este se repite 60 veces por segundo.

Ahora algunos tecnicismos: un trazo del haz de la pantalla se conoce como 'línea de barrido horizontal'. La línea de barrido horizontal es la unidad de medida fundamental para la distancia vertical en la pantalla. Se especifica la altura de una imagen, indicando el número de líneas horizontales que abarca. El período durante el cual el haz regresa del extremo derecho al extremo izquierdo se conoce con el nombre de 'borrado horizontal'. El período durante el cual el haz

regresa a la parte alta de la pantalla se conoce como 'borrado vertical'. El proceso completo de trazar una pantalla toma 16.684 microsegundos. El perìodo de borrado vertical es alrededor de 1.400 microsegundos. El borrado horizontal demora 14 microsegundos. Una lìnea horizontal corresponde a 64 microsegundos.

La mayorìa de los televisores se diseñan con 'sobrebarrido'; esto significa que ellos amplìan la imagen hasta un extremo tal que sus bordes exceden los lìmites del tubo de televisión. Esto da la seguridad de que no habrá bordes desagradables en la imagen. Sin embargo, constituye un problema para los computadores, porque naturalmente la información, que cae fuera de los bordes de la pantalla, se pierde; por lo tanto, el cuadro que despliega el computador, debe ser algo menor que lo que teoricamente puede desplegar una pantalla de televisión. Por esta razón, el despliegue Atari emplea normalmente sólo 192 lìneas horizontales; asì el lìmite normal de resolución de un televisor usado con un computador Atari es de 192 pixels en el sentido vertical. La unidad standard de distancia horizontal es el 'perìodo o compás de color'. Se especifica el ancho de una imagen, indicando cuantos perìodos de color comprende. Hay 228 perìodos de color en una lìnea de barrido horizontal, de los cuales un máximo de 176 son visibles. Asì, el lìmite máximo de resolución horizontal de color con un receptor de color normal es de 176 pixels. Con el computador personal Atari es posible lograr mayor fineza y controlar individualmente los semi-perìodos. Esto da una resolución horizontal de 352 pixels. Sin embargo, el empleo de esta capacidad dará origen a efectos de color muy interesantes conocidos como artificios de color. Los artificios de color pueden ser una molestia cuando no se les desea; pueden ser también la delicia del programadoe que desea colores adicionales y no se deja amedrentar por sus restricciones.

COMPUTADORES Y TELEVISORES

El problema fundamental que tiene todo microcomputador, al usar un sistema de barrido de televisión para propósito de despliegue, es que el despliegue de televisión es un proceso dinámico, ya que el televisor no tiene memoria de su imagen. En consecuencia, el computador debe recordarla constantemente y volver a enviar la señal al televisor, indicándole lo que debe desplegar. Este proceso de envìo de información al televisor es de tipo continuo y requiere atención permanente. Por esta razón, la mayorìa de los microcomputadores tienen circuitos especìficos para entenderse con el televisor. El arreglo básico es virtualmente el mismo en todos los sistemas:

microprocesador---)RAM de pantalla---)circuito de video---)pantalla de televisión

El microprocesador escribe la información a la RAM de pantalla que contiene los datos de pantalla. Los circuitos de video constantemente están interrogando esta área de RAM, obteniendo los datos de pantalla y los convierten en señales de televisión. Estas señales van al televisor, que a continuación las despliega. La memoria de pantalla se transfiere a la pantalla en el mismo orden en que se encuentra en la RAM, es decir, el primer byte de la memoria de pantalla se transfiere al rincón superior izquierdo de la pantalla. El segundo byte toma la posición inmediatamente a la derecha, a continuación el tercero, el cuarto y así sucesivamente hasta el último byte, que se ubicará en el rincón inferior derecho de la pantalla.

La calidad de la imagen que se obtiene sobre la pantalla dependerá de dos factores: la calidad de la circuitería de video y la cantidad de RAM de video usada para el despliegue. El arreglo más simple es el usado por TRS-80 y PET. (TRS-80 es marca registrada de Radio Schak Co.; PET es marca registrada de Commodore Business Machines). Estos dos productos asignan un Kbyte de RAM específico como memoria de pantalla. Los circuitos de video simplemente sacan los datos de esa área, la interpretan como caracteres, usando un juego de caracteres en ROM y ponen los caracteres resultantes en la pantalla. Cada byte representa un carácter, lo que permite una elección de 256 diferentes caracteres en el juego. Con 1 Kbyte de RAM de pantalla pueden desplegarse mil caracteres. No es mucho lo que se puede lograr con este arreglo. Apple emplea circuitos de video más avanzados. (Apple es marca registrada de Apple Computers). Se proveen aquí tres modos gráficos: texto, gráfico de baja resolución: lo-res, y gráfico de alta resolución: hi-res. El modo gráfico de texto funciona practicamente en la misma forma en que lo hacen PET y TRS-80. En el modo gráfico de baja resolución, los circuitos de video llegan a la memoria de pantalla y la interpretan en forma diferente. En vez de interpretar a cada byte como un carácter, cada byte se interpreta como un par de cuaternas de color. El valor de cada cuaterna especifica el color de un solo pixel. En el modo de alta resolución cada bit en la memoria de pantalla corresponde a un pixel. Si el bit está puesto, el pixel tiene color; si el bit es 0, el pixel permanece obscuro. La situación se complica por la gran variedad de diseños de Apple, pero permanece siempre la idea básica. La idea importante es que Apple tiene tres modos de despliegue, tres formas diferentes de interpretar los datos de la memoria de pantalla. Los circuitos de video de Apple son suficientemente hábiles como para interpretar un byte de memoria de pantalla, ya sea como un carácter de 8 bit (en el modo de texto), dos cuaternas de color de 4 bits c/u (modo de

baja resolución), u 8 bits individuales para un mapa de bit (modo de alta resolución).

ANTIC, UN MICROPROCESADOR DE VIDEO

El sistema de lista de despliegue ATARI constituye una generalización de estos tres sistemas. En lugar del modo único de PET y TRS-80 y de los tres modos de Apple, ATARI tiene 15. La segunda gran diferencia está en que los diferentes modos de despliegue pueden combinarse en la pantalla. Es decir, el usuario no está restringido a la elección entre una pantalla de texto o una pantalla gráfica. Cualquier combinación de los 15 modos gráficos puede desplegarse de una vez en la pantalla. La tercera diferencia de importancia es que la memoria de pantalla puede ubicarse en cualquier parte en el espacio de direccionamiento del computador y moverse en él mientras corre el programa, mientras que en las otras máquinas se trata de áreas de RAM de pantalla fijas.

Esta generalización es posible gracias a un microprocesador de video llamado ANTIC. Mientras los primeros sistemas usaban circuitería de video relativamente sencilla, Atari diseñó un microprocesador completo, solamente para preocuparse de las complicaciones del despliegue de televisión. ANTIC es todo un microprocesador, tiene su juego de instrucciones, un programa y datos. El programa para ANTIC recibe el nombre de 'lista de despliegue'. La lista de despliegue especifica tres cosas: dónde se encuentran los datos de pantalla, qué modos de despliegue deben usarse para =interpretar estos datos de pantalla, y qué opción de despliegue especial (si es que hay alguna) debería implementarse. Al usar las listas de despliegue, es importante olvidarse de la antigua imagen de una pantalla como un todo homogéneo en un solo modo y verlo en cambio como un apilamiento de 'líneas de modo'. Una línea de modo es un conjunto de líneas de barrido horizontal. Se extiende horizontalmente a través de toda la pantalla. Una línea del modo gráfico 2 comprende 16 líneas de barrido horizontales, mientras que una línea del modo gráfico 7 mide solamente 2 líneas de barrido de alto. Muchos de los modos gráficos disponibles en BASIC son homogéneos. Toda la pantalla se monta en un solo modo; sin embargo uno no debe limitar su imaginación a este esquema, con la lista de despliegue puede crearse cualquier secuencia de líneas de modo a lo alto de la pantalla. La lista de despliegue es un conjunto de bytes de código que especifica esta secuencia.

El conjunto de instrucciones de ANTIC es bastante simple. Existen cuatro clases de instrucciones: instrucciones de modo de mapa (gráfica), instrucciones de modo de

caracteres, instrucciones de líneas en blanco e instrucciones de salto. Las instrucciones de modo de mapa hacen que ANTIC despliegue una línea de modo con pixels simples de color (no caracteres). Las instrucciones de modo de caracteres hacen que ANTIC despliegue una línea de modo con caracteres. Las instrucciones de línea en blanco hacen que ANTIC despliegue una cantidad de líneas de barrido horizontal en el color parejo del fondo. Las instrucciones de salto son similares a las instrucciones JMP de un 6502, ellas recargan el contador de programa de ANTIC. Tambien existen cuatro opciones especiales que pueden especificarse ocasionalmente para poner un bit determinado en la instrucción de ANTIC. Estas opciones son: interrupción de lista de despliegue (DLI), carga de exploración de memoria (LMS), movimiento vertical y movimiento horizontal (scroll).

Con las instrucciones del modo de mapa, ANTIC despliega la línea de modo conteniendo pixels con colores sólidos en cada uno de ellos. El color desplegado proviene de un registro de color. La elección del registro de color se especifica por medio del valor del dato de pantalla. En los modos de gráfica o mapa de cuatro colores (BASIC 3, 5, 7 y 15; ANTIC 8, A, D y E), se requiere un par de bits para especificar un color:

valor del par de bits		registro de color empleado
00	0	COLBAK
01	1	COLPF0
10	2	COLPF1
11	3	COLPF2

Como sólo se requieren dos bits para especificar un pixel, 4 pixels se codifican en cada byte de datos de pantalla. Por ejemplo, un byte de dato de pantalla que tenga el valor hexadecimal 1B desplegaría 4 pixels; el primero tendría el color de fondo, el segundo el del registro de color 0, el tercero el del registro 1 y el cuarto el del registro 2:

$1B = 00011011 = 00 01 10 11

En los modos de mapa de dos colores (BASIC 4, 6, 8 y 14; ANTIC 9, B, C y F) cada bit especifica uno de dos registros de color. Un bit 0 selecciona el color de fondo para el pixel y un bit 1 selecciona el registro de color 0 para el pixel. Se almacenan así ocho pixels en un byte de datos de pantalla.

ANTIC tiene ocho diferentes modos de despliegue de mapa. Difieren en el número de colores que contienen (2 o 4), la

altura vertical ocupada por una línea de modo (1, 2, 4 u 8 líneas de barrido), y el número de pixels que caben horizontalmente en una línea de modo (40, 80, 160 o 320). Así algunos modos de mapa dan mejor resolución; éstos naturalmente requerirán mayor cantidad de RAM de pantalla. La Figura 2.1 presenta esta información para todos los modos.

Los modos Basic del nueve al once son modos GTIA, no disponibles con CTIA, el antiguo circuito de interfaz de TV; los modos GTIA corresponden a modo ANTIC F; para mayores detalles vea el apéndice X.

QE

modo GTIA	modo ANTIC	modo BASIC	colo- res	l.barr. /l.modo	pixels /l.modo	bytes /línea	bytes /pant.
0	2	0	2	8	40	40	960
0	3	-	2	10	40	40	760
0	4	12	4	8	40	40	960
0	5	13	4	16	40	40	480
0	6	1	5	8	20	20	480
0	7	2	5	16	20	20	240
0	8	3	4	8	40	10	240
0	9	4	2	4	80	10	480
0	A	5	4	4	80	20	960
0	B	6	2	2	160	20	1920
0	C	14	2	1	160	20	3840
0	D	7	4	2	160	40	3840
0	E	15	4	1	160	40	7680
0	F	8	2	1	320	40	7680
1	F	9	1	1	80	40	7680
2	F	10	9	1	80	40	7680
3	F	11	16	1	80	40	7680

Figura 2.1
Los modos ANTIC, GTIA y BASIC y sus requerimientos de líneas.

Las instrucciones de modos de caracteres hacen que ANTIC despliegue una línea de modo con caracteres. Cada byte en la RAM de pantalla especifica un carácter. Existen seis modos de despliegue de caracteres. Los despliegues de caracteres se discuten el el capítulo 3.

Las instrucciones de línea en blanco producen líneas en blanco en el color uniforme del fondo. Existen ocho instrucciones de línea en blanco; especifican desde una hasta 8 líneas de barrido en blanco.

Hay dos instrucciones de salto. La primera (JMP) es un salto directo; recarga el contador de programa de ANTIC con una nueva dirección que sigue como operando a la instrucción JMP. Su única función es proveer una solución a un problema intrincado: El contador de programa ANTIC solamente tiene diez bits de contador y seis bits enclavados, de manera que la lista de despliegue no puede cruzar límites de 1 Kbyte. Si la lista de despliegue debe cruzar un límite de 1 Kbyte, entonces debe usarse una instrucción JMP para saltar por sobre ese límite. Note que esto significa que las listas de despliegue no son totalmente reubicables.

La segunda instrucción de salto (JVB) es la que más se usa. Recarga el contador de programa con el valor del operando y espera que el televisor realice el borrado vertical. Esta instrucción se usa normalmente para terminar con una lista de despliegue, saltando hacia atrás hasta su mismo comienzo. Retornando así hasta el comienzo de una lista de despliegue, la transforma en un bucle infinito; la espera del borrado vertical asegura que este bucle infinito siempre esté sincronizado con el ciclo de despliegue del televisor. Tanto JMP como JVB son instrucciones de tres bytes; el primer byte corresponde al código operacional, los segundo y tercer bytes corresponden a la dirección a la cual debe saltarse (byte menos significativo, después byte más significativo).

Las cuatro opciones especiales mencionadas previamente, se discutirán en los capítulos 5 y 6. La opción de carga de exploración de memoria (LMS) debe tener una explicación preliminar. Se elige esta opción poniendo el bit 6 del byte de instrucción de un modo de mapa o de carácter. Cuando ANTIC encuentra una instrucción de este tipo, cargará su contador de exploración de memoria con los siguientes dos bytes. Este contador de exploración de memoria le indica a ANTIC donde está la memoria de pantalla. Comenzará a traer datos de despliegue de esa área. La instrucción LMS es una instrucción de tres bytes: un byte de código operacional seguido de dos bytes de operando. En listas de despliegue simples la instrucción LMS se usa una sola vez, al comienzo de la lista. A veces puede ser necesaria una segunda instrucción LMS.
La necesidad nace cuando el área de RAM de pantalla supera una barrera de 4 Kbytes. El contador de exploración de memoria solamente tiene 12 bits de contador y 4 bits enclavados; así los datos de despliegue no pueden cruzar una barrera de 4 Kbytes. En este caso debe usarse una instrucción LMS para saltar con el contador de exploración de memoria por sobre este límite. Note que esto significa que los datos de despliegue no son totalmente reubicables. Las instrucciones LMS tienen usos más amplios, que se discutirán más adelante.

LA CONSTRUCCION DE LISTAS DE DESPLIEGUE

Toda lista de despliegue debe comenzar con 3 instrucciones de 8 líneas en blanco. Esto hace bajar el despliegue en 24 líneas de barrido, con lo cual se anula el posible efecto de un sobrebarrido vertical. Después de esto, debería especificarse la primera línea de despliegue. Simultáneamente debe usarse LMS para indicar a ANTIC donde encontrará la memoria de pantalla. Sigue a continuación la lista de despliegue propiamente tal, que lista los bytes de despliegue para las líneas de modo de la pantalla. El número total de líneas de barrido horizontales producidas por la lista de despliegue siempre debería ser 192 o menos; ANTIC no mantiene los requerimientos de tiempo del televisor. Si ANTIC tiene demasiadas líneas de barrido en su despliegue, lo realizará, pero la imagen del televisor probablemente perderá sincronismo. Desplegando menos de 192 líneas de barrido no causará problemas; de hecho disminuirá el tiempo de ejecución del 6502, al reducir el número de ciclos desviados por ANTIC. El programador debe calcular la suma de líneas de barrido horizontal producidas por su lista de despliegue y verificarla. La lista de despliegue termina con una instrucción JVB. He aquí una lista de despliegue típica para un despliegue normal de modo gráfico 0 BASIC (todos los valores se dan como hexadecimales):

```
70    8 líneas en blanco
70    8 líneas en blanco
70    8 líneas en blanco
42    modo de despliegue ANTIC 2 (modo 0 BASIC)
20    y la memoria de pantalla comienza en 7C20
7C
02    despliegue de modo ANTIC 2
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
41    salto y espera del borrado vertical
E0    a la lista de despliegue que se inicia en
7B    $7BE0
```

Como se puede ver, la lista de despliegue es corta --- solamente 32 bytes. La mayoría de las listas de despliegue tienen menos de 100 bytes. Más encima, son muy simples en su estructura y fáciles de construir.

Para implementar su propia lista de despliegue, primero debe diseñar el formato del despliegue. La mejor manera de hacerlo es sobre un papel. Trace la imagen de la pantalla y tradúzcala en una secuencia de líneas de modo. Lleve cuenta de las líneas de barrido de su despliegue, mirando los requerimientos de líneas de barrido de los varios modos según Figura 2.1. Traduzca la secuencia de líneas de modo en una secuencia de bytes de modo ANTIC. Anteponga tres bytes ($70) de ocho líneas en blanco al comienzo de la lista. Ponga el bit número 6 del primer byte de despliegue, (es decir haga la cuaterna superior igual a cuatro). Esto constituye un comando de exploración de carga de memoria. Continúe con dos bytes que

especifiquen la dirección de la RAM de pantalla (byte menos significativo, después byte más significativo). Continúe con el resto de los bytes de despliegue. Al terminar su lista de despliegue coloque la instrucción JVB ($41) y la dirección de comienzo de la lista de despliegue. Ahora almacene todos estos bytes en RAM. Pueden estar en cualquier lugar; sólo asegúrese que no se sobrepongan en alguna forma a otros datos y que su JVB apunte al comienzo de la lista de despliegue. La lista de despliegue no debe cruzar límites de 1 Kbyte. Si es absolutamente indispensable cruzar un límite de este tipo, inserte una instrucción JMP justamente antes de este límite. El operando de la instrucción JMP naturalmente será la dirección del primer byte al otro lado de este límite. A continuación debe detener a ANTIC por una fracción de segundo mientras reescribe su puntero de lista de despliegue. Hágalo poniendo un 0 en SDMCTL en la ubicación $22F. A continuación almacene la dirección de la nueva lista de despliegue en $230 y $231 (primero byte menos significativo, después byte más significativo). Por último reconecte ANTIC con un $22 en SDMCTL. Durante el intervalo de borrado vertical mientras ANTIC este quieta, el Sistema Operativo recargará el contador de programa de ANTIC con estos valores.

ESCRIBIENDO EN LA PANTALLA DE UNA LISTA DE DESPLIEGUE CONSTRUIDA

La memoria de pantalla puede ubicarse en cualquier parte del espacio de direccionamiento del computador. Normalmente la lista de despliegue especifica el comienzo de la memoria de pantalla con la primera instrucción de despliegue --- la instrucción LMS inicial. Sin embargo, si se desea, ANTIC puede ejecutar una nueva instrucción LMS con cada nueva línea de la lista de despliegue. Así puede desplegarse en una sola pantalla información distribuida a través de todo el espacio de direccionamiento del computador. Esto puede ser útil al disponer ventanas de texto independientes.

Hay varias restricciones en la ubicación de la memoria de pantalla. Primero, la memoria de pantalla no puede cruzar límites de 4 Kbytes de direccionamiento. Si no puede evitar cruzar límites de 4 K (como sería el caso en los modos BASIC 8, 9 10 u 11 que usan alrededor de 8 Kbyte de RAM), debe recargarse el contador de exploración de memoria con una nueva instrucción LMS. Segundo, si Ud. desea usar algunas de las rutinas de pantalla del Sistema Operativo, debe seguir las convenciones que usa el Sistema Operativo. Esto puede ser particularmente difícil al usar una lista de despliegue modificada en un programa BASIC. Si Ud. altera la lista de despliegue normal de un programa BASIC y a continuación intenta hacer un PRINT o PLOT en la pantalla, el Sistema

Operativo lo hará bajo la asunción de que la lista de despliegue no ha sido modificada. Probablemente se producirá un despliegue confuso.

Hay tres causas por las cuales el despliegue puede fallar cuando Ud. intente esto. Primero, BASIC puede rehusarse a ejecutar la operación de pantalla porque ella constituye un imposible en el modo gráfico que el Sistema Operativo piensa que existe. El Sistema Operativo almacena el valor del modo gráfico en el cual cree que se encuentra la pantalla, en la dirección $57. Ud. puede engañar el Sistema Operativo y obtener su cooperación, haciendo el POKE de un valor nuevo aquì. Hasa POKE del número de modo BASIC, no del número de modo ANTIC.

El segundo problema puede producirse cuando Ud. mezcle lìneas con diferentes requerimientos de bytes de memoria de pantalla. Algunos modos requieren 40 bytes por lìnea, otros requieren 20 bytes y algunos solamente 10. Supongamos que Ud. inserta una lìnea de modo de 20 bytes en una lista de despliegue que tiene lìneas de modo de 40 bytes. Enseguida Ud. hace un PRINT al texto del despliegue. Todo es normal antes de llegar a la lìnea singular, pero bajo ella los caracteres están desplazados en 20 espacios hacia la derecha. Ello es debido a que el Sistema Operativo supuso que cada lìnea requerirìa 40 bytes y ubicó consecuentemente los caracteres, pero ANTIC cuando se encontró con la lìnea singular tomó solamente 20 bytes, de lo que el Sistema Operativo pensó que serìa una lìnea de 40. ANTIC interpretó los otro 20 bytes como pertenecientes a la lìnea siguiente y los desplegó ahì. A consecuencia de ello, la lìnea siguiente y todas las posteriores estarán desplazadas en 20 espacios hacia la derecha.

La única forma absoluta para evitar este problema es retraerse del uso de PRINT y PLOT en BASIC, al escribir a una pantalla de lista de despliegue construida. La solución rápida y sucia es organizar la pantalla en grupos de lìneas que contengan múltiplos ìntegros del requerimiento de bytes normal; es decir: no inserte lìneas de modo de 20 bytes en un despliegue de 40 bytes por lìnea; inserte siempre 2 lìneas de 20 bytes o una de 20 y 2 de 10. Mientras Ud. mantenga los múltiplos ìntegros apropiados, el desplazamiento horizontal podrá evitarse.

Esta solución destaca el tercer problema entre las listas de despliegue mezcladas y BASIC: el desplazamiento vertical. El Sistema Operativo ubica el material de pantalla verticalmente, calculando el número de bytes que debe saltarse desde la parte de arriba de la pantalla. En un despliegue

normal de 40 bytes por línea, BASIC ubicaría los caracteres de la décima línea saltando 360 bytes desde el comienzo. Si Ud. ha insertado 4 líneas de 10 bytes, BASIC aterrizará en la pantalla 3 líneas más abajo de lo que podría haberse esperado. Además los diferentes modos consumen diferente cantidad de líneas de barrido, de manera que la posición en la pantalla no será exactamente la esperada, a no ser que se tome en cuenta el costo en líneas de barrido en cada caso.

Se ve que, emplear modos mixtos puede resultar difícil en conjunto con el Sistema Operativo. Frecuentemente debe engañarse al Sistema Operativo para hacer funcionar despliegues de este tipo. Para hacer PRINT o PLOT en una ventana de modo, haga un POKE del número de modo BASIC de esa ventana a la dirección $57, enseguida un POKE de la dirección del pixel del rincón superior izquierdo de la ventana de modo a las ubicaciones $58 y $59 (primero byte menos significativo después byte más significativo). En modos de carácter, ejecute un POSITION 0,0 para reponer en su origen el cursor en el rincón superior izquierdo de la ventana de texto. En modos gráficos de mapa, todos los PLOT y los DRAWTO serán usando el rincón superior izquierdo de la ventana de modo como origen del sistema de coordenadas.

El sistema de lista de despliegue puede usarse para lograr despliegues de pantalla muy atractivos. El uso más obvio es el de la mezcla de texto con gráficos. Por ejemplo, Ud. podría preparar una pantalla con un título en letra gruesa del modo 2 BASIC, un subtítulo de tamaño mediano en el modo BASIC 1 y una impresión fina general del modo BASIC 0. A continuación podría disponer una figura en el modo BASIC 8 al centro de la pantalla con algo más de texto en la parte de abajo. Un buen ejemplo de esta técnica se da en el despliegue del programa de Estados y Capitales.

Los problemas descritos impiden el uso extensivo de estas técnicas desde BASIC. Con rutinas de lenguaje Assembler, las listas de despliegue modificadas se usan en la forma más conveniente, organizando la pantalla en una serie de ventanas, en que cada ventana tiene su propia instrucción LMS y su propia área de RAM independiente.

APLICACIONES DE LAS LISTAS DE DESPLIEGUE

Una aplicación simple de las modificaciones de listas de despliegue es el desplazamiento vertical de las líneas de la pantalla insertando bytes de líneas en blanco. Ello agregará algún espaciamiento vertical que permite destacar mensajes críticos y mejorar la legibilidad de algunos despliegues.

Otro uso importante de las manipulaciones de listas de despliegue lo constituye el acceso a características no disponibles desde BASIC. Ello es especialmente cierto en los modelos 400 y 800. En ellos existen dos modos de texto soportados por ANTIC , pero no por BASIC. Estos mismos modos son accesibles como modos 12 y 13 en los modelos XL.
En estos casos y él del modo ANTIC 3, solamente manipulaciones de listas de despliegue permiten al usuario acceder a estos modos. También hay capacidades de interrupción de lista de despliegue y de movimiento fino que solamente pueden lograrse después de haber modificado la lista de despliegue. Estas características constituyen los objetivos de los capítulos 5 y 6.

Las manipulaciones con la instrucción LMS y su operando ofrecen muchas posibilidades al programador creativo. Por ejemplo, cambiar el LMS durante el borrado vertical, permite al programador alternar imágenes de diferentes pantallas. Ello se puede hacer a baja velocidad para alternar entre despliegues pretrazados sin tener que reconstruir cada uno. Cada despliegue permanecerá en (y consumirá) RAM aunque no esté en uso, pero estaría disponible casi instantaneamente. La misma técnica puede usarse también para animación. Saltando a través de una secuencia de despliegues puede lograrse una animación cíclica. El programa que lo haga, solamente tendrá que manipular dos bytes de dirección para desplegar miles de bytes de RAM.

También es posible superponer imágenes alternando pantallas a alta velocidad. El ojo humano tiene un tiempo de resolución de alrededor de 1/16 de segundo, de manera que un programa puede ciclar entre cuatro imágenes, una cada 1/60 de segundo y así repetir el conjunto cada 1/15 de segundo. De esta forma, hasta cuatro imágenes pueden residir aparentemente en forma simultánea en la pantalla.
Naturalmente, existen algunas desventajas en este método. Primero, cuatro imágenes separadas requieren una gran cantidad de RAM. Segundo, cada imagen de despliegue aparecerá deslavada porque sólo está presente 1/4 del tiempo. Esto significa que el

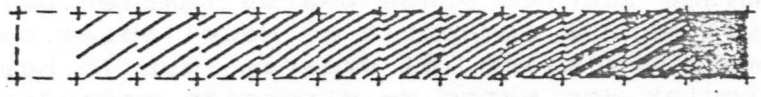
fondo de todos los despliegues debe ser negro y cada imagen debe ser brillante. Además habrá algún parpadeo de pantalla desagradable cuando se use esta técnica. Un programador conservador podría más bien considerar ciclar entre solamente dos o tres imágenes a la vez. Esta técnica también puede usarse para extender la resolución de color y luminosidad del computador, ciclando entre cuatro versiones de la misma imagen, cada versión mostrando un cierto rango de luminosidad o de color, permitirá un rango de colores y luminosidades total más amplio. Por ejemplo, suponga que queremos desplegar una barra de muchas luminancias diferentes. Primero ponemos nuestros registros de color en los siguientes valores:

```
Fondo  :  00
campo 1:  02
campo 2:  0A
campo 3:  0C
```

Ahora colocamos las siguientes imágenes en cada una de las áreas de RAM de pantalla:

	contenido del pixel (según reg.color)											
primer cuadro	1	1	1	1	2	3	2	3	2	3	2	3
segundo cuadro	F	1	1	1	F	F	2	3	2	3	2	3
tercer cuadro	F	F	1	1	F	F	F	F	2	3	2	3
cuarto cuadro	F	F	F	1	F	F	F	F	F	F	2	3
luminancia efectiva x4	2	4	6	8	10	12	20	24	30	36	40	48

luminancia percibida



En esta forma es posible lograr una resolución de luminancia mucho más fina.

Una sugestión final se refiere a un punto cargado de oportunidades pero poco comprendido hasta el momento: la lista de despliegue dinámica. Es ésta una lista de despliegue que el 6502 modifica durante los perìodos de borrado vertical. Deberìa ser posible producir efectos interesantes con listas de despliegue dinámicas. Por ejemplo, un programa de edición de texto dinamicamente inserta lìneas en blanco sobre y bajo la

línea de pantalla que se está editando, para separarla del resto de las líneas del texto. A medida que el cursor se mueve verticalmente, la lista de despliegue se altera. La técnica es compleja pero muy efectiva.

CAPITULO 3
INDIRECCION GRAFICA (REGISTROS DE COLOR Y JUEGOS DE CARACTERES)

La indirección es un concepto poderoso en la computación, pero un tanto difícil de captar para el programador novicio. En el lenguaje ensamblador para 6502 hay tres niveles de indirección en relación a los números. El primero y más directo es el modo de direccionamiento inmediato, en el cual se especifica directamente el número propiamente tal:

```
LDA #$F4
```

El segundo nivel de indirección se alcanza al referirse el programa a la ubicación de memoria que contiene el número:

```
LDA $0602
```

El tercer nivel de indirección y el más alto en el 6502, se alcanza cuando el programa se refiere a un par de ubicaciones de memoria, las cuales en conjunto contienen la dirección de memoria que a su vez contiene el número. En el 6502 esta indirección se complica por la adición de un índice:

```
LDA ($D0), Y
```

La indirección provee al programador con un mayor nivel de generalidad (y por ende de poder). En vez de sacar a relucir siempre el mismo número, cada vez que uno quiere algo de él, el programador simplemente puede apuntarlo. Cambiando el puntero puede cambiar el comportamiento del programa. Obviamente la indirección es una capacidad muy importante.

La indirección gráfica está incluída en el Sistema de Computación Personal ATARI en dos formas: a través de los registros de color y a través de los juegos de caracteres. Los programadores que se familiarizan inicialmente con este computador después de haber programado otros sistemas, muchas veces piensan en términos de colores directos. Un registro de color es algo más complejo que un color. Un color especifica un valor permanente. Un registro de color es indirecto, contiene cualquier valor de color. La diferencia entre los dos es análoga a la diferencia entre una llave de corona y una llaves de dados. La llave de corona tiene un tamaño específico,

en cambio la llave de dados puede sosportar casi cualquier tamaño de dado. Una llave de dados es más flexible, pero requiere un poco más de habilidad para utilizarla bien. En la misma forma, un registro de color es más flexible que un color, pero requiere mayor habilidad para usarlo en forma eficaz.

Los computadores ATARI tienen nueve registros de color; cuatro se emplean para los gráficos de jugadores-proyectiles y se discutirán en el capìtulo 4. Los cinco remanentes no siempre se usan en su totalidad; dependiendo del modo gráfico en uso, puede que se usen tan pocos como 2 o tantos como 5 en una misma pantalla. En el modo BASIC 0 solamente se emplea un registro y medio debido a que el valor de color de los caracteres se ignora; los caracteres tienen el mismo color del registro de campo 2, pero obtienen su luminosidad del registro 1. Los registros de color se encuentran en la direcciones $D01E hasta $D01A en CTIA o GTIA, según el caso. Son copiados desde las ubicaciones RAM del Sistema Operativo a la CTIA o GTIA durante el intervalo de borrado vertical. La Figura 3.1 da las direcciones de circuito y de copia de los registros de color.

IMAGEN	CIRCUITO		REG. COPIA	
CONTROLADA	NOMBRE	DIRECCION	NOMBRE	DIRECCION
jugador 0	COLPM0	D012	PCOLR0	2C0
jugador 1	COLPM1	D013	PCOLR1	2C1
jugador 2	COLPM2	D014	PCOLR2	2C2
jugador 3	COLPM3	D015	PCOLR3	2C3
campo 0	COLPF0	D016	COLOR0	2C4
campo 1	COLPF1	D017	COLOR1	2C5
campo 2	COLPF2	D018	COLOR2	2C6
campo 3	COLPF3	D019	COLOR3	2C7
fondo	COLBK	D014	COLOR4	2C8

Figura 3.1
nombres y direcciones de los registros de color

Para la mayorìa de los propósitos, el usuario controla los registros de color escribiendo a las ubicaciones de copia. Hay solamente dos casos en los cuales el programador escribirá directamente a las direcciones CTIA/GTIA. El primero es el más común; corresponde a interrupciones de lista de despliegue, las que serán vistas en el Capìtulo 5. El segundo se presenta cuando el usuario inhibe las rutinas de interrupción del borrado vertical del Sistema Operativo, que mueven los valores de copia hasta CTIA/GTIA. Las interrupciones del borrado vertical se analizan en el apéndice I.

Los colores se codifican en los registros de color por medio de una fórmula muy simple. La cuaterna superior da el matiz del color que es idéntico al segundo parámetro del comando BASIC SETCOLOR. La tabla 9.3 del Manual de Referencia BASIC lista los valores de matiz. La cuaterna inferior del registro de color da el valor de luminosidad del color. Es el mismo del tercer parámetro del comando BASIC SETCOLOR. El bit de menor orden de esta cuaterna no tiene importancia, así existen ocho luminosidades para cada matiz. Por lo tanto existen 128 colores para elegir (8 luminosidades por 16 matices). En este libro el término color indicará siempre una combinación matiz-luminosidad.

Una vez que un color se ha codificado en un registro, se le transfiere a la pantalla haciendo referencia al registro de color que lo contiene. En los modos de despliegue de mapa o gráficos que permiten cuatro registros de color, los datos de pantalla especifican cual de los registros debe hacerse corresponder a cada punto en la pantalla. Como existen cuatro registros de color, se requieren solamente dos bits para codificar cada pixel. Así, cada byte de datos de pantalla contiene cuatro pixels. El valor de cada par de bits especifica qué registro de color es el que provee el color para ese pixel.

En los modos de despliegue de texto BASIC 1 y 2, la selección del registro de color se efectúa a través de los dos bits de orden superior del código del carácter. Esto naturalmente deja sólo 6 bits para la definición del carácter propiamente tal, por lo cual en estos dos modos solamente existen 64 caracteres.

La indirección de color le da al programador cuatro capacidades especiales. Primero, el programador puede escoger entre 128 colores diferentes para sus despliegues. Esto le permitirá escoger el color que más se acerque a sus necesidades.

Segundo, el programador puede manipular los registros de color en tiempo real para producir efectos muy hermosos. Su versión más simple se puede demostrar por la siguiente línea BASIC:

```
FOR I=0 TO 254 STEP 2:POKE 712,I:NEXT I
```

Esto simplemente hace ciclar el color del borde a través de todos los colores posibles. El efecto es muy agradable y

ciertamente atrae la atención. Esta técnica fundamental puede extenderse en una variedad de vías. Una variación especial es crear simple animación cíclica trazando una figura en cuatro colores y enseguida ciclando los colores a través de los registros más que redibujando la figura. El siguiente programa ilustra la idea:

```
10 GRAPHICS 23
20 FOR X=0 TO 39
30 FOR I=0 TO 3
40 COLOR I
50 PLOT 4*X+I,0
60 DRAWTO 4*X+I,95
70 NEXT I
80 NEXT X
90 A=PEEK(712)
100 POKE 712,PEEK(710)
110 POKE 710,PEEK(709)
120 POKE 709,PEEK(708)
130 POKE 708,A
140 GOTO 90
```

La tercera aplicación de los registros de color es asignar colores en forma lógica de acuerdo a situaciones. Por ejemplo, un sistema de menú por página puede hacerse más comprensible haciendo cambiar el color de fondo o del borde de acuerdo a la página del menú. Talvez la pantalla podría dar un destello rojo al presionar una tecla ilegal. El uso de los caracteres en colores, disponibles en los modos gráficos BASIC 1 y 2 puede en gran medida aumentar el impacto de un material de texto. La suma de una cuenta podría mostrarse en rojo si la cuenta realmente se encuentra en el rojo, o en negro si el saldo es positivo. Palabras o frases de importancia pueden desplegarse en colores especiales para hacerlas destacarse. El uso de los colores en los modos de mapas o gráficos (no textos) también puede mejorar la utilidad de esta gráfica. Una simple imagen gráfica (un monstruo, un bote o cualquier otra cosa) puede presentarse en colores diferentes para representar diferentes versiones de lo mismo. Es muy costoso en términos de RAM almacenar una imagen, pero cuesta muy poco cambiar el color de una imagen ya existente. Por ejemplo, sería mucho más fácil mostrar tres botes distintos presentando la misma forma en tres colores diferentes, que mostrar tres formas diferentes.

La cuarta y más importante aplicación de los registros de color se utiliza con las interrupciones de lista de despliegue. Un solo registro de color puede utilizarse para poner hasta 128 colores en la pantalla. Esta importantísima

posibilidad se discutirá en el capìtulo 5.

JUEGOS DE CARACTERES

La indirección gráfica también se presenta a través del juego de caracteres redefinible. Hay un juego de caracteres normales en ROM, pero no hay razón para tener que usar necesariamente ese juego. El usuario puede crear y desplegar todo tipo de carácter que desee. Hay tres pasos que son necesarios para usar un juego de caracteres redefinido. Primero, el programador debe definir el juego de caracteres. Este es el paso que más tiempo requiere. Cada carácter se despliega en la pantalla sobre una red de 8x8 y está codificado en la memoria como una tabla de 8 bytes. La Figura 3.2 ilustra este arreglo de codificación.

Imagen del carácter	rep. binaria	rep. hexadecimal
. . . . .		
. . . . .	00000000	00
. . . .X.X. . . .	00011000	18
. . .X.X.X.X. . .	00111100	3C
. .X.X. . .X.X. .	01100110	66
. .X.X. . .X.X. .	01100110	66
. .X.X.X.X.X.X. .	01111110	7E
. .X.X. . .X.X. .	01100110	66
. . . . .	00000000	00

Figura 3.2
Codificacion de caracteres

Un juego completo contiene 128 caracteres, cada uno de ellos en versión normal y de video inverso. Un juego de caracteres de este tipo requiere 1024 bytes de espacio de memoria y debe iniciarse en un lìmite de 1K. Los juegos de caracteres para los modos BASIC 1 y 2 solamente poseen 64 caracteres diferentes, por lo que requieren solamente 512 bytes y deben iniciarse en un lìmite de un 1/2K. Los ocho primeros bytes definen el carácter de orden 0, los siguientes ocho él de orden 1 y asì sucesivamente. Obviamente, el definir un nuevo juego de caracteres es un trabajo pesado. Afortunadamente existen en el mercado paquetes de software que facilitan esta labor.

Una vez que el juego de caracteres se ha definido y

puesto en RAM, debe indicarse a ANTIC dónde encontrar este juego. Ello se logra haciendo un POKE del número de página de comienzo de la tabla de caracteres en la ubicación $D409 (decimal 54281). La ubicación de copia del Sistema Operativo, que es la ubicación que uno normalmente utiliza, se llama CHBAS y está en el $2F4 (decimal 756). El tercer paso en el uso de un juego de caracteres es la impresión del carácter sobre la pantalla. Esto se hace directamente desde BASIC con PRINT o escribiendo su número de orden en forma directa a la memoria de pantalla.

Una capacidad especial del sistema, soportada en BASIC solamente en los modelos XL, es la opción de caracteres a cuatro colores. Los modos gráficos BASIC 1 y 2 soportan 5 colores, pero cada carácter, dentro de estos modos, de hecho es solamente un carácter bicolor. Cada uno de ellos tiene solamente un color de fondo y un color de carácter. El color de carácter puede ser uno de cuatro, pero solamente uno a la vez puede mostrarse dentro de un carácter determinado. Esto puede constituir una seria limitación cuando se esté trabajando con gráfica de caracteres. Existen otros dos modos de texto especialmente aptos para gráfica de caracteres. Ellos son los modos ANTIC 4 y 5 y corresponden a los modos BASIC 12 y 13 en los modelos XL. En estos modos cada carácter solamente tiene un ancho de cuatro pixels, pero cada pixel puede tener uno de cuatro colores (incluyendo él de fondo). Los caracteres están definidos igual como en el modo gráfico BASIC 0, con excepción de que cada pixel tiene el doble de ancho y tiene 2 bits asignados para especificar el registro de color que le corresponde. A diferencia de los modos ANTIC 6 y 7 (modos BASIC 1 y 2), la selección de los registros de color no se hace por el nombre del carácter sino que por el juego de caracteres definido. Cada byte en la tabla del carácter se conforma a través de 4 pares de bits, cada uno de los cuales elige el color correspondiente a un pixel. (Es por esto que solamente existen cuatro pixels horizontales por carácter). El bit más alto (D7) del byte del nombre del carácter modifica el registro de color a usar. La selección del registro de color se hace de acuerdo a la Figura 3.3:

par de bits en la def. del carácter	D7 = 0	D7 = 1
00	COLBAK	COLBAK
01	campo 0 PF0	PF0 campo 0
10	campo 1 PF1	PF1 campo 1
11	campo 2 PF2	PF3 campo 3

Figura 3.3
Selección del registro de color para caracteres

Usando estos modos de texto, pueden ponerse caracteres gráficos multicolores en la pantalla.

Otro modo de caracteres ANTIC de interés, es el modo de minúsculas con bajada (modo 3 ANTIC). Este modo despliega 10 líneas de barrido por línea de modo, pero dado que los caracteres usan solamente 8 bytes en sentido vertical, las dos líneas de barrido inferiores normalmente están desocupadas. Si se despliega un carácter del último cuarto del juego de caracteres, las dos líneas de barrido superiores de este carácter permanecerán desocupadas; los datos que deberían haberse desplegado allí, aparecerán en cambio en las dos líneas inferiores. Esto permite al usuario crear caracteres minúsculas con bajada.

De la indirección de los juegos de caracteres nace una serie de otras posibles aplicaciones muy interesantes y útiles. Una aplicación obvia es la modificación de la tipografía. Esto puede dar a un programa una apariencia realmente única. También es posible tener juegos de caracteres de letras griegas, cirílicas u otros de tipo especial. Yendo un paso más allá, pueden crearse juegos de caracteres para gráficos. El programa computacional ENERGY CZAR (ENERGIA) emplea un juego de caracteres redefinidos para sus gráficos de barras. Cada carácter tiene un alto de 8 pixels; esto significa que un gráfico de barras implementado con caracteres normales tiene una resolución de 8 pixels, lo cual es bastante pobre. ENERGY CZAR usa un juego de caracteres especiales, en el cual algunos de los símbolos de texto menos usados han sido reemplazados por caracteres especiales de gráficos de barra. Un carácter corresponde a una barra de un pixel, otro a una barra de dos pixels y así sucesivamente hasta llegar a la barra de ocho pixels. Así el programa puede trazar gráficos de barra con resolución de hasta un pixel. La Figura 3.4 muestra un despliegue típico de este programa. La mezcla de texto con gráfica de tipo mapa solamente es aparente; en realidad todo el despliegue es construído con caracteres.

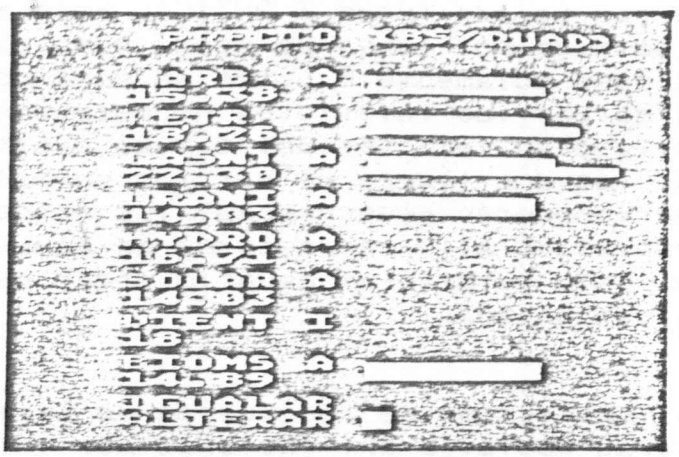


Figura 3.4
Gráfico de barras de ENERGY CZAR

En muchas aplicaciones pueden crearse caracteres que poseen imágenes especiales. Por ejemplo, definiendo un juego de caracteres de gráfica de terreno con caracteres de rìo, caracteres de árboles, caracteres de montaña y asì sucesivamente, es posible hacer un mapa de terreno de cualquier paìs. De hecho, con imaginación, un mapa de terreno de un planeta diferente puede hacerse con igual facilidad. Al hacerlo lo mejor es definir 5 u ocho caracteres para cada tipo de terreno. Cada variación dentro de un tipo deberìa ubicarse en una posición levemente diferente en cuanto al pixel del carácter. Mezclando los diferentes caracteres uno con otro, es posible evitar la apriencia monótona que es caracterìstica de un juego de caracteres gráficos muy primitivos. La mayor parte de la gente no se dará cuenta que el mapa resultante usa gráfica de caracteres hasta que realicen un estudio cuidadoso del mapa. La Figura 3.5 muestra el despliegue de un mapa de terreno creado con un juego de caracteres gráficos. La reproducción en blanco y negro no hace justicia al despliegue original, que corresponde al juego de estrategia 'Eastern Front', y que tiene hasta 18 colores.

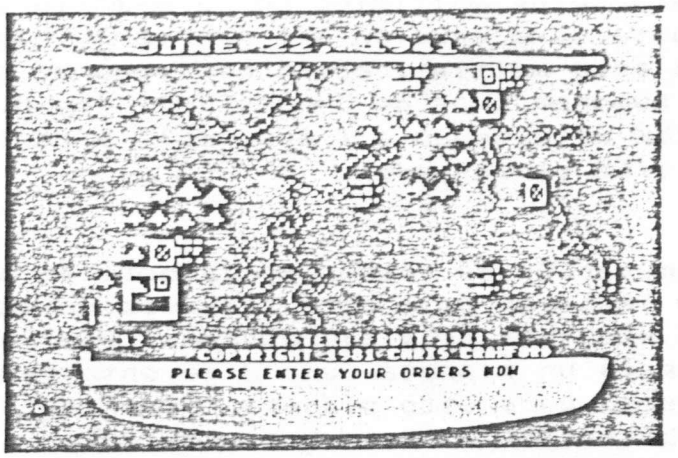


Figura 3.5
Mapa de terreno con juego de caracteres gráficos

Puede crearse también un juego de caracteres electrónicos con caracteres de transistor, caracteres de diodo, caracteres de conductores y así sucesivamente para producir todo un programa de trazado de esquemáticos electrónicos. Podría también crearse un juego de caracteres de arquitectura con caracteres de puerta, de muro, de rincón y así sucesivamente hasta tener un programa completo de planos arquitónicos. Las posibilidades gráficas que se abren con la gráfica de caracteres en los computadores personales, hasta el momento no se han explorado en su plenitud.

Los caracteres pueden ponerse de cabeza haciendo POKE 4 en la ubicación 755 decimal.
Un aprovechamiento posible de esta característica sería el despliegue de las cartas del naipe (como por ejemplo en un juego de 21 real). La mitad superior de una carta puede ponerse al derecho; con una interrupción de lista de despliegue los caracteres pueden ponerse de cabeza para la

mitad inferior de la carta. Esta característica también podría ser de interés para desplegar imágenes en reflexiones especulares (piscinas, lagos, etc).

Posibilidades aún más excitantes nacen cuando uno se da cuenta que es bastante práctico cambiar el juego de caracteres durante el transcurso del programa.
Un juego de caracteres cuesta ya sea 512 bytes o 1024 bytes; en cualquier caso es un costo bastante bajo para mantener un juego de caracteres múltiples en la memoria y saltar entre uno y otro durante la ejecución del programa. Hay tres regímenes de tiempo para esta multiplicación de juego de caracteres: lento para seres humanos (más de 1 segundo); rápido para seres humanos (1/60 de segundo a 1 segundo); y rápido de máquina (más rápido que 1/60 de segundo).

La conmutación de juegos de caracteres a velocidad humanamente lenta es útil para "cambios de escenario". Por ejemplo, un programa de viaje espacial podría usar un juego de caracteres para un planeta, uno diferente para el espacio, y un tercer juego para otro planeta. A medida que el viajero cambia de ubicación, el programa cambia los juegos de caracteres, para dar origen a nuevos y exóticos escenarios. Un programa del tipo aventura podría cambiar juegos de caracteres a medida que los jugadores cambien de lugares.

La conmutación de juegos de caracteres a velocidades humanamente rápidas, se usa primariamente para efectos de animación. Esto puede hacerse en dos formas: cambiando los caracteres dentro de un mismo juego de caracteres y cambiando el juego de caracteres como un conjunto. SPACE INVADERS (marca registrada de Taito America Corp.) en los ATARI 400/800 usa la primera técnica. Los invasores de hecho son caracteres. Cambiándolos rapidamente, el programador fue capaz de animarlos. Fue fácil porque existen solamente 6 monstruos diferentes. Cada uno tiene cuatro encarnaciones distintas.
Es posible una animación cíclica de alta velocidad sobre toda la pantalla, estableciendo un sistema de juegos de caracteres, trazando la imagen en la pantalla y enseguida simplemente ciclando a través de los juegos de caracteres. Si cada carácter tiene una encarnación levemente diferente en cada juego de caracteres, el carácter atravesará una secuencia de animación a medida que los juegos de caracteres cambien. Por esta vía una pantalla llena de objetos puede hacerse mover cíclicamente mediante un bucle muy sencillo. Una vez que los datos del juego de caracteres estén en su lugar y la pantalla ha sido trazada, la rutina para animar la pantalla podría ser tan simple como esto:

```
1000 FOR I=1 TO 10
1010 POKE 756,BASECAR(I)
1020 NEXT I
1030 GOTO 1000
```

La animación con juego de caracteres a velocidad de computador se usa para poner varios juegos de caracteres en una sola pantalla. Esto hace uso de interrupciones de lista de despliegue en el computador, tópico que se ampliará en el Capìtulo 5.

El uso de juego de caracteres para gráfica y animación tiene muchas ventajas y algunas limitaciones. La mayor ventaja es que cuesta muy poca RAM el producir despliegues detallados. Un despliegue gráfico que use caracteres del modo BASIC 2 (tal como el que se muestra en la Figura 3.5) puede dar tanto o más detalle como un despliegue del modo BASIC 7, y aún un color adicional, a pesar de que la imagen de caracteres costará tan poco como 200 bytes mientras la imagen tipo mapa tendrá un costo de 4000 bytes. El costo en RAM para juegos de caracteres múltiples es solamente de 512 bytes por juego, de manera que es bastante barato tener juegos de caracteres múltiples. La manipulación de pantalla con gráfica de caracteres es mucho mas rápida porque se trata de manipular menos datos; sin embargo la gráfica de caracteres no es tan flexible como la de mapa. No es posible poner cualquier cosa en cualquier lugar de la pantalla. Esta limitación podrìa impedir el uso de la gráfica de caracteres en algunas aplicaciones; sin embargo quedan muchas aplicaciones gráficas para las cuales los programas necesitan despliegues que tienen solamente una cantidad limitada de figuras predefinidas en ubicaciones determinadas. En estos casos las gráficas de caracteres ofrecen una gran utilidad.

CAPITULO 4
GRAFICA DE JUGADORES-PROYECTILES

La animación es una capacidad importante en todos los sistemas de computación personal. La actividad sobre la pantalla puede agregar mucho al éxito y realismo de cualquier programa. Ciertamente la animación es crucial como atractivo de muchos juegos computacionales.
Más importante, una imagen animada puede transmitir información con más impacto y mayor claridad que una imagen estática. Puede llamar la atención sobre una cosa o un hecho de importancia. Puede mostrar directamente un proceso dinámico más que indirectamente referirse a él. La animación por lo tanto debe mirarse como un elemento importante en las capacidades gráficas de cualquier sistema de computación.

La forma convencional de realizar animación en un computador personal es mover los datos de la imagen a través del área RAM de la pantalla. Esto requiere un proceso de dos pasos. Primero, el programa debe borrar la imagen antigua escribiendo los valores de fondo sobre la RAM que contenga la imagen actual. A continuación el programa debe escribir los datos de la imagen en la RAM que corresponda a la nueva posición de ella. Repitiendo una y otra vez este proceso, la imagen aparentemente se mueve a través de la pantalla.

Hay dos problemas asociados con esta técnica. Primero, si la animación se realiza en un modo gráfico de pixels de gran tamaño, el movimiento no será continuo; la imagen saltará a través de la pantalla. Con otros computadores, la única solución es usar un modo gráfico con pixels de menor tamaño (resolución más alta).
El segundo problema es mucho más grave. La pantalla es una imagen bi-dimensional, pero la memoria de pantalla está organizada uni-dimensionalmente. Esto significa que una imagen con elementos contiguos en la pantalla, no los tendrá en forma contigua en la RAM. Esta discrepancia se ilustra en la Figura 4.1.

```
          IMAGEN                    Bytes
                                    en RAM
.........................
.........................   00  00  00
.........X..XX..X.........   00  99  00
.........X.XXXX.X.........   00  BD  00
.........XXXXXXXX.........   00  FF  00
.........X.XXXX.X.........   00  BD  00
.........X..XX..X.........   00  99  00
.........................   00  00  00

          distribución de los bytes en RAM:

00 00 00 00 99 00 00 BD 00 00 FF 00 00 BD 00 00 99 00 00 00 00
            ↑        ↑        ↑        ↑        ↑
            +--------+--------+--------+--------+
                              |

     los bytes de la imagen están dispersos a través de la RAM
```

Figura 4.1
Las imágenes en RAM no son contiguas

La importancia de esta discrepancia no se hace evidente hasta que uno no trate de escribir un programa para producir un movimiento de imagen de este tipo. Vea como los bytes que conforman la imagen están dispersos a través de la RAM. Para borrarlos, el programa primero debe calcular sus direcciones. Este cálculo no siempre es fácil. El código assembler, que accede un solo byte en la ubicación de pantalla (XPOS, YPOS) tendría la siguiente apariencia (el programa supone 40 bytes por cada línea de pantalla):

```
LDA SCRNRM      Dirección del comienzo de la RAM de pantalla
STA POINTR      puntero página cero
LDA SCRNRM+1    byte superior de la direccion
STA POINTR+1    byte superior del puntero
LDA #$00
STA TEMPA+1     registro temporal
LDA YPOS        posición vertical
ASL A           por 2
ROL TEMPA+1     desplazar "carry" a TEMPA+1
ASL A           por 4
ROL TEMPA+1     nuevo desplazamiento
ASL A           por 8
ROL TEMPA+1     nuevo desplazamiento
LDX TEMPA+1     almacenar YPOS*8
STX TEMPB+1     en TEMPB
STA TEMB        byte inferior
ASL A           por 16
ROL TEMPA+1
ASL A           por 32
ROL TEMPA+1
CLC
ADC TEMPB       sumar en YPOS*8 para tener YPOS*40
STA TEMPB
LDA TEMPA+1     ahora el byte superior
ADC TEMPB+1
STA TEMPB+1
LDA TEMPB       TEMPB contiene el desplazamiento desde el
                comienzo de la pantalla hasta el pixel.
CLC
ADC POINTR
STA POINTR
LDA TEMPB+1
ADC POINTR+1
STA POINTR+1
LDY XPOS
LDA (POINTR), Y
```

Obviamente, esta rutina para acceder a una ubicación de pantalla es demasiado compleja. No es ciertamente la forma más rápida y elegante para resolver el problema; con seguridad un buen programador podria sacar ventaja de ciertas circunstancias especiales para hacer el código más compacto o elegante. Lo importante de todo esto es que acceder los pixels de la pantalla da un montón de trabajo computacional. La rutina de más arriba demora alrededor de 100 ciclos de máquina para acceder un solo byte de la pantalla. Mover una imagen que ocupa, digamos, unos 50 bytes, requeriría 100 accesos o 10.000 ciclos de máquina, lo que es del orden de 10 milisegundos. Esto puede no parecer mucho, pero si quiere conseguirse un movimiento suave debe moverse el objeto cada

17 milisegundos. Si hay otros objetos que deban moverse u otros cálculos que hacer, entonces no queda mucho tiempo de procesador para dedicarle. En conjunto esto significa que el tipo de animación descrito (llamado animación de campo) es demasiado lento para la mayoria de los propósitos. Puede conseguirse animación asi, pero hay una limitación a objetos pequeños o movimiento lento con algunos cálculos entre uno y otro. Las concesiones que el programador debe hacer para poder usar este tipo de animación son demasiado restrictivas.

La solución ATARI a este problema es la gráfica de jugador-proyectil. Para entender la gráfica jugador-proyectil es importante entender la esencia del problema de la animación de campo: la pantalla es bidimensional mientras que la RAM es unidimensional. La solución fue crear un objeto gráfico que fuera unidimensional en la pantalla como también unidimensional en RAM. Este objeto, llamado jugador, aparece en RAM como una tabla numérica que tiene ya sea 128 o 256 bytes de largo. La tabla se transfiere directamente a la pantalla. Aparece como una cinta vertical que se extiende desde arriba hasta la base de la pantalla. Cada byte de la tabla tiene su correspondencia ya sea en una o dos lineas de barrido. La elección entre ambas alternativas la toma el programador. La imagen de pantalla es un simple mapa de bits de los datos de la tabla. Si el bit está puesto, entonces en el pixel correspondiente de la columna vertical hay imagen; si el bit no existe, es igual a 0, el pixel correspondiente no da imagen. Asi, la imagen del jugador no es estrictamente unidimensional, de hecho tiene 8 bits de ancho.

Trazar la imagen de un jugador sobre la pantalla es sumamente sencillo. Primero se traza una figura de la imagen deseada sobre papel cuadriculado. Esta imagen debe tener no más de 8 pixels de ancho. A continuación se traduce esta imagen a código binario, sustiyendo por valor 1 los pixels iluminados y por 0 los pixels obscuros. Enseguida se traduce el número binario resultante a decimal o hexadecimal, dependiendo de cual de ambos resulta más conveniente. Enseguida, se almacenan ceros en la RAM del jugador para limpiar la imagen; a continuación se guardan los datos de la imagen en la memoria RAM del jugador; el byte de la parte más alta del jugador va primero, seguido por los otros bytes de imagen en secuencia de arriba hacia abajo. Mientras más abajo en la RAM Ud. ubique los datos, más abajo en la pantalla aparecerá la imagen.

Animar esta imagen es muy simple. El movimiento vertical se obtiene moviendo los datos de la imagen a través de la RAM del jugador. En principio éste es el mismo método usado en la animación de campo, pero hay una gran diferencia en la práctica: la rutina de movimiento para el desplazamiento vertical es de movimiento unidimensional en vez de

bidimensional. El programa no tiene necesidad de multiplicar por 40 y muchas veces ni siquiera requiere de la indirección. Podria ser tan simple como esto:

```
        LDX #$01
BUCLE-LDA JUGADOR,X
      STA JUGADOR-1,X
      INX
      BNE BUCLE
```

Esta rutina requiere alrededor de 4 milisegundos para mover todo el jugador, aproximadamente la mitad de lo requerido por la animación de campo, que solamente mueve unos 50 bytes, mientras que ésta mueve 256. Si se requiere alta velocidad, el bucle puede arreglarse para que solamente mueva los bytes de la imagen propiamente tal y no todo el jugador; bajo estas condiciones el bucle correria facilmente en unos 100 a 200 micro-segundos. Lo importante es que el movimiento vertical de jugadores es más simple y más rápido que el movimiento de objetos de campo.

El movimiento horizontal es aún más fácil que el vertical. Hay un registro de jugador llamado el registro de posición horizontal. El valor de este registro ubica la posición horizontal del jugador sobre la pantalla. Todo lo que hay que hacer es poner un número en este registro para que el jugador salte a esa posición horizontal. Para mover al jugador horizontalmente simplemente cambie el número almacenado en el registro de posición horizontal. No hay nada más que hacer.

Los movimientos vertical y horizontal son independientes; pueden combinarse de la forma que uno quiera.

La escala del registro de posición horizontal es de un ciclo de reloj por unidad. Asi, agregar uno al registro de posición horizontal moverá al jugador en un compás de reloj hacia la derecha. Solamente hay 228 compases de reloj en una línea de barrido; además algunos de estos no se despliegan porque corresponden a sobrebarrido. El registro de posición horizontal puede contener 256 posiciones; algunas de estas están fuera del borde izquierdo o derecho de la pantalla. Dependiendo del sobrebarrido del televisor, las posiciones de 0 a 44 estarán más allá del borde izquierdo de la pantalla y de las posiciones 220 a 255 estarán más allá del borde derecho de la pantalla. Así, la región visible del jugador corresponde a las posiciones horizontales desde 44 hasta 220. Recuerde sin embargo, que esto puede variar de televisor a televisor; un rango conservador sería de 60 a 200. Este rango de coordenadas a veces puede ser un tanto incómodo de usar, pero ofrece una

característica simpática: una forma muy simple de sacar a un jugador de la pantalla es poner su posición horizontal en 0. Con un simple load o store en ensamblador (o un POKE en BASIC), el jugador desaparecerá.

El sistema descrito hasta aquí permite la realización de animación a alta velocidad. Existen adicionalmente una serie de embellecimientos que en gran medida aumentan la utilidad. El primer embellecimiento es que existen cuatro jugadores independientes. Todos estos jugadores tienen su propio juego de registros de control y áreas de RAM, de modo que su operación sea absolutamente independiente. Se les llama P0 a P3. Pueden ser usados uno al lado del otro para dar una resolución horizontal de 32 bits o usarse independientemente para permitir cuatro objetos móviles. Cada jugador tiene su propio registro de color, este registro es absolutamente independiente del registro de color del campo. Los registros de color de jugadores se llaman COLP (X) y tienen su copia en PCOLR(X). Esto da la posibilidad de poner mucho color en la pantalla. Sin embargo cada jugador tiene un solo color; no es posible obtener jugadores multicolores sin interrupciones de lista de despliegue. (Las interrupciones de listas de despliegue se discuten en el Capítulo 5). Cada jugador tiene ancho controlado. Puede ser puesto en normal, doble o cuádruple ancho con los registros SIZEP(X). Esto es útil para hacer tomar a los jugadores diferentes tamaños. También puede escogerse la resolución vertical de los jugadores. Puede usarse resolución de línea simple, caso en el cual cada byte de la tabla de un jugador ocupa una línea de barrido horizontal, o resolución de doble línea. En este caso usa dos líneas de barrido horizontales. Con resolución de una línea, la tabla del mapa de bits de cada jugador tiene 256 bytes de largo, con doble línea cada tabla es de 128 bytes. Es éste el único aspecto en que las propiedades de los jugadores no son independientes. La selección de resolución vertical se aplica a todos ellos. La resolución vertical de los jugadores se controla por el bit D4 del registro DMACTL. En resolución de doble línea, más o menos los primeros 10 bytes de la tabla del jugador se pierden debido al sobrebarrido vertical y caen fuera del borde de la pantalla. Los últimos veinte bytes se pierden debido a que caen fuera del borde inferior de la pantalla. En resolución simple, se pierden veinte y cuarenta bytes aproximadamente por esta razón.

La siguiente exquisitez es la provisión de proyectiles. Estos son objetos gráficos de 2 bits de ancho asociados a los jugadores. Hay un proyectil asignado a cada jugador. El color que toma es él del registro de color de su jugador. Los datos de forma del proyectil provienen de la tabla del mapa de bits de proyectil en RAM justo por delante de las tablas de jugadores. Los cuatro proyectiles están empaquetados en la misma tabla. Cuatro proyectiles por dos bits por proyectil da

ocho bits. Los proyectiles pueden moverse independientemente de los jugadores. Tienen sus propios registros de posición horizontal. Los proyectiles tienen su propio registro de tamaño, SIZEM, que puede disponer el tamaño horizontal en la misma forma en que lo hace SIZEP(X) para los jugadores. Sin embargo, los proyectiles no pueden tener diferentes tamaños, todos se ponen al mismo tiempo. Los proyectiles son útiles como balas o como líneas delgadas verticales sobre la pantalla. Si se desea, los proyectiles pueden agruparse para formar un quinto jugador, cuyo color en ese caso es él del registro de color del campo de juego 3. Ello se logra poniendo el bit D4 del registro de prioridad de control (PRIOR). Note que los proyectiles aún con esta opción en efecto, pueden moverse independientemente, ya que sus posiciones horizontales están establecidas por sus regsitros de posición horizontal. El bit de habilitación de quinto jugador solamente afecta el color de los proyectiles.

El movimiento vertical del proyectil se realiza en la misma forma que un jugador: moviendo los datos de la imagen del proyectil a través del área de RAM del proyectil. Esto puede ser difícil, ya que los proyectiles están agrupados dentro de la misma tabla en la RAM. Para acceder a un proyectil determinado, deben enmascararse los bits correspondientes a los demás.

Una característica importante de la gráfica de jugador-proyectil es que tanto jugadores como proyectiles son absolutamente independientes del campo. Pueden mezclarse con cualquier modo gráfico, sea de texto o de mapas. Esto presenta un problema: qué pasa si un jugador queda sobre la imagen de un campo? Cuál imagen tiene prioridad? Existe la opción de definir las prioridades que se usan al desplegar jugadores. Si quiere, todos los jugadores pueden tener prioridad sobre todos los registros de color de campo, o puede imponer que todos los registros de color de campo (excepto él de fondo), tengan prioridad sobre todos los jugadores o puede poner que los jugadores 0 y 1, de aquí en adelante referidos como P0 y P1, tengan prioridad sobre todos los registros de campo de juego, con P2 y P3 a su vez teniendo menor prioridad que el campo, o puede poner el registro de color del campo 0 y el del campo 1 (PF0 y PF1) para que tengan prioridad sobre todos los jugadores, los que a su vez tendrán prioridad sobre PF2 y PF3. Estas prioridades se seleccionan con el registro de control de prioridad (PRIOR) que tiene su copiador en GPRIOR. Esta capacidad permite al jugador pasar por delante de una imagen y detrás de otra, dando lugar así a efectos tridimensionales.

El último toque es la provisión de detección de colisión. Ella tiene valor especialmente para juegos. Ud. puede controlar si algún objeto gráfico (jugador o proyectil) ha chocado con alguna otra cosa. Específicamente puede verificar colisiones entre proyectiles y jugadores, campos de juego y proyectiles, entre jugadores y entre campo y jugador. Existen 54 colisiones posibles; cada una de ellas tiene asignado un bit que puede verificarse. Si el bit está puesto, ha ocurrido esa colisión. Estos bits están ubicados en 15 registros de GTIA (solamente los 4 bits de menor orden se usan; incluso algunos no tienen valor). Se trata de registros de lectura solamente; no pueden reponerse escribiendo Ø en ellos. Los registros pueden limpiarse para una detección de colisión posterior, escribiendo cualquier valor en el registro HITCLR. Mediante este comando todos los registros de colisión quedan limpios.

En términos circuitales, se produce una colisión cuando la imagen de un jugador coincide con otra imagen; así el bit de colisión no será puesto hasta que en la pantalla se trace la parte de la imagen que corresponda a la colisión. Esto significa que la detección de colisión puede no ocurrir hasta 16 milésimas de segundo después de haberse movido el jugador. La solución preferida a este problema es ejecutar el movimiento del jugador y la detección de colisión durante la rutina de interrupción del intervalo de borrado vertical (vea el Apéndice I). En este caso la detección de colisión debería verificarse primero, a continuación despejar la colisión y mover los jugadores. Otra solución es esperar por lo menos 16 milisegundos después de mover a un jugador y antes de verificar cualquier colisión que envuelva a ese jugador.

Hay una serie de pasos necesarios para utilizar gráfica de jugadores-proyectiles. Primero debe separarse un área de RAM para jugadores-proyectiles e indicar al computador donde se encuentra; si se usa resolución de una línea esta RAM corresponderá a 1280 bytes; si se usa resolución de doble línea tendrá un largo de 640 bytes. Es una buena práctica usar el área que está justamente ante el área de despliegue en la parte alta de la RAM. La distribución del área de jugador-proyectil se muestra en la Figura 4.2

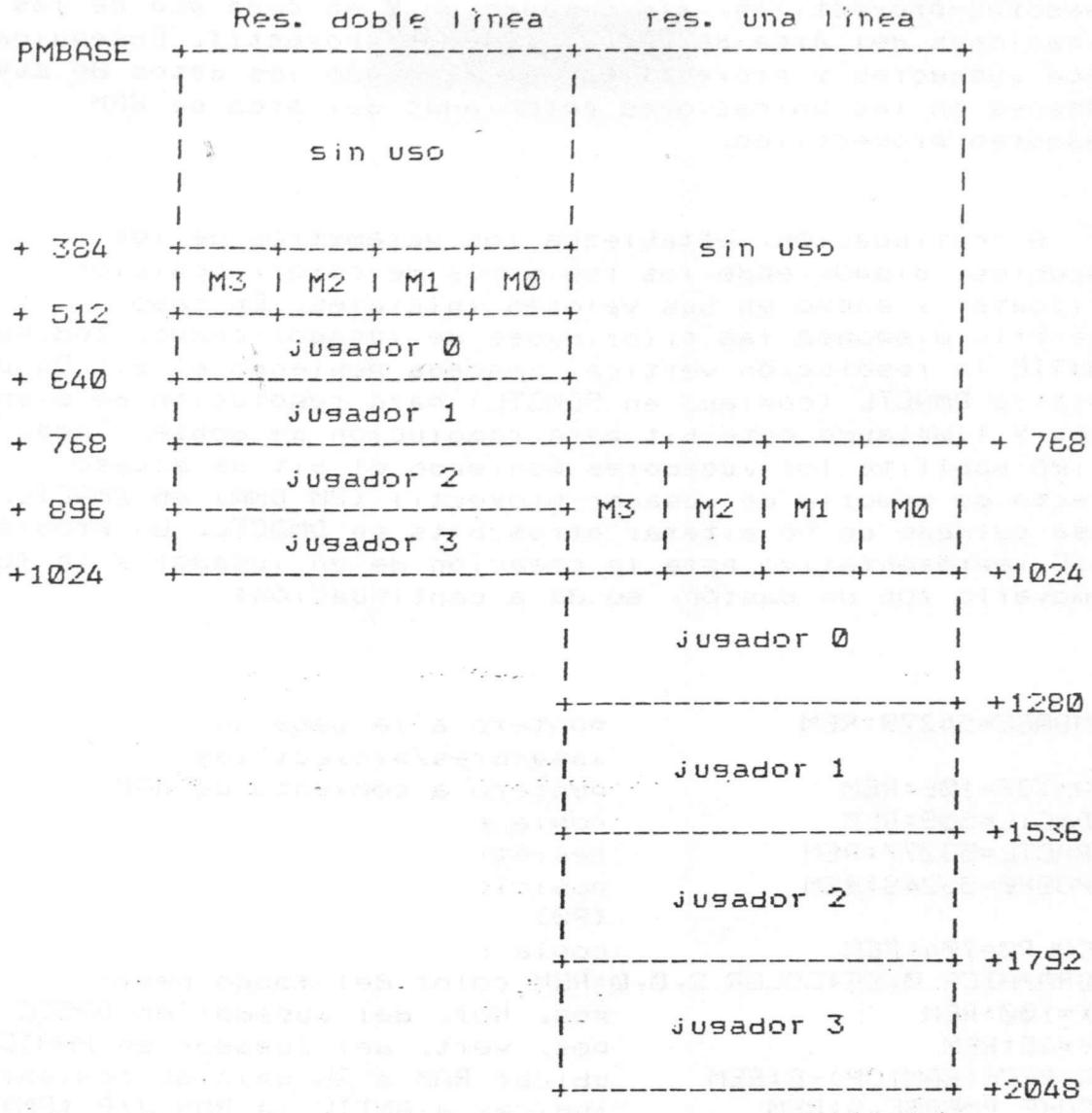


Figura 4.2
Distribución del área de RAM de jugadores-proyectiles

El puntero que indica el comienzo del área jugadores-proyectiles se llama PMBASE. Debido a las limitaciones internas de ANTIC, PMBASE debe estar en un límite de 1 KByte para resolución de simple línea o de 2 KByte para resolución de doble línea. Si Ud. decide no usar a todos los jugadores o ninguno de los proyectiles, las áreas de RAM separadas para los objetos no usados pueden emplearse para otros propósitos. Una vez decidido donde se encontrará el área de RAM de jugadores-proyectiles, debe indicarse esto a ANTIC, almacenando el número de página de PMBASE en el registro PMBASE de ANTIC.

El paso siguiente es limpiar la RAM de jugadores-proyectiles, almacenando un 0 en cada una de las ubicaciones del área de RAM de jugador-proyectil. Enseguida trace jugadores y proyectiles, almacenando los datos de sus imágenes en las ubicaciones apropiadas del área de RAM jugadores-proyectiles.

A continuación, establezca los parámetros de los jugadores, disponiendo los registros de color, posición horizontal y ancho en sus valores iniciales. En caso necesario disponga las prioridades de jugador-campo. Indique a ANTIC la resolución vertical deseada poniendo el bit D4 del registro DMACTL (copiado en SDMCTL) para resolución de simple línea y limpiando este bit para resolución de doble línea. Por último habilite los jugadores poniendo el bit de acceso directo de memoria de jugador-proyectil (PM DMA) en DMACTL. Tenga cuidado de no alterar otros bits en DMACTL. Un programa BASIC representativo para la creación de un jugador y la forma de moverlo con un bastón, se da a continuación:

```
1 PMBASE=54279:REM          puntero a la base de
                            jugadores/proyectiles
2 RAMTOP=106:REM            puntero a comienzo de RAM
3 SDMCTL=559:REM            copia en RAM del registro DMACTL
4 GRACTL=53277:REM          registro de control grafico GTIA
5 HPOSP0=53248:REM          posicion horizontal del jugador 0
                            (P0)
6 PCOLR0=704:REM            copia del color de P0
10 GRAPHICS 0:SETCOLOR 2,0,0:REM color del fondo negro
20 X=100:REM                pos. hor. del jugador en BASIC
30 Y=48:REM                 pos. vert. del jugador en BASIC
40 A=PEEK(RAMTOP)-8:REM     ubicar RAM a 2K bajo su comienzo
50 POKE PMBASE,A:REM        indicar a ANTIC la RAM J/P (PM)
60 MIBASEJP=256*A:REM       direccion RAM jugador/proyectil
70 POKE SDMCTL,46:REM       habilitar acceso directo a
                            memoria (DMA) para J/P (PM) con
                            resolucion de doble línea
80 POKE GRACTL,3:REM        habilitar despliegue PM
90 POKE HPOSP0,100:REM      dar posicion horizontal
100 FOR I=MIBASEJP+512 TO MIBASEJP+640:REM este bucle borra
                            el jugador
110 POKE I,0
120 NEXT I
130 FOR I=MIBASEJP+512+Y TO MIBASEJP+518+Y
140 READ A:REM              este bucle dibuja el jugador
150 POKE I,A
160 NEXT I
170 DATA 8,17,35,255,32,16,8
180 POKE PCOLR0,88:REM      jugador rosado
```

```
190 A=STICK(0):REM          lectura bastón
200 IF A=15 THEN GOTO 190:REM si no hay movimiento, preguntar
                            de nuevo
210 IF A=11 THEN X=X-1:POKE HPOSP0,X
220 IF A=7 THEN X=X+1:POKE HPOSP0,X
230 IF A<>13 THEN GOTO 280
240 FOR I=8 TO 0 STEP -1
250 POKE MIBASEJP+512+Y+I,PEEK(MIBASEJP+511+Y+I)
260 NEXT I
270 Y=Y+1
280 IF A<>14 THEN GOTO 190
290 FOR I=0 TO 8
300 POKE MIBASEJP+511+Y+I,PEEK(MIBASEJP+512+Y+I)
310 NEXT I
320 Y=Y-1
330 GOTO 190
```

Una vez desplegados los jugadores, puede resultar difícil eliminarlos de la pantalla. Esto se debe a que el procedimiento mediante el cual se les despliega envuelve varios pasos. Primero, ANTIC encuentra los datos de jugador-proyectil desde la RAM (si esta búsqueda se habilita en DMACTL). A continuación ANTIC transfiere los datos de jugador-proyectil a GTIA (siempre que esta acción esté habilitada por GRACTL). GTIA a su vez despliega todo lo que se encuentre en sus registros gráficos de jugadores y proyectiles (GRAFP0 hasta GRAFP3 y GRAFM). Muchos programadores tratan de eliminar la gráfica de jugador-proyecil limpiando los bits de control en DMACTL y GRACTL. Esto sólo evita que ANTIC envíe nuevos datos de jugador-proyectil a GTIA; los datos anteriores que se encuentren en los registros GRAF(X) seguirán siendo desplegados. Para despejar totalmente los jugadores deben limpiarse los registros GRAF(X) una vez que los bits de control de DMACTL y GRACTL hayan sido puestos en 0. Una solución más simple es dejar dispuesto el jugador pero poner su posición horizontal en 0. Naturalmente, usando esta solución, ANTIC continuará recurriendo al acceso directo de memoria para reponer los datos de jugador-proyectil, gastando alrededor de 10.000 ciclos de máquina por segundo.

La gráfica de jugador-proyectil ofrece una serie de capacidades muy especiales. Naturalmente es de gran valor en animaciones. En todo caso tiene algunas limitaciones: solamente existen cuatro jugadores y cada uno de ellos solamente mide ocho bits de ancho. Si Ud. necesita más bits de resolución horizontal siempre puede recurrirse a la animación de campo. Pero para animación de gran velocidad y que no requiera gran definición, la gráfica de jugador-proyectil es la mejor solución.

Es posible escribir directamente los datos en los registros gráficos del jugador-proyectil (GRAFP(X)) de GTIA sin pasar por ANTIC. Esto otorga al programador un control más directo sobre la gráfica de jugador-proyectil. También incrementa su responsabilidad correspondientemente. El programador debe mantener un mapa de bits de los datos del jugador-proyectil y traspasarlos a los registros gráficos en el momento oportuno. En consecuencia el 6502 debe ser encadenado al ciclo de trazado de pantalla. (Vea la discusión sobre núcleos en el Capítulo 5). Es esta una técnica compleja que ofrece algunas mejoras de comportamiento a cambio de un gran esfuerzo de programación. El programador que no recurra al potencial circuital de ANTIC, debe reemplazarlo forzosamente con su sudor.

La gráfica de jugador-proyectil tambien ofrece muchas capacidades fuera de la animación. Los jugadores son una excelente forma para aumentar la cantidad de colores en un despliegue. Los cuatro registros de color adicionales que proveen, permiten de hecho cuatro colores más sobre cada línea del despliegue. Naturalmente, la resolución de ocho bits limita el rango de aplicación. Hay sin embargo una forma en que se puede evitar esta limitación en algunas ocasiones. Defina un jugador de cuádruple ancho y póngalo sobre la pantalla. A continuación disponga las prioridades, de modo que el jugador tenga una prioridad menor que el campo. A continuación invierta los colores del campo y del fondo, de modo que el aparente color de fondo en la pantalla realmente corresponde a color de campo. El jugador desaparecerá tras este fondo simulado. Corte ahora un hoyo en este fondo simulado, trazando el verdadero fondo sobre él. El jugador aparecerá por sobre el verdadero color de fondo, pero solamente en el área donde se ha trazado este verdadero color de fondo. En esta forma, el jugador puede tener más de ocho bits de resolución horizontal. Un programa ejemplo para esto sigue a continuación:

```
1 RAMTOP=106:REM          Puntero comienzo RAM
2 PMBASE=54279:REM        Puntero RAM jugador/proyectil
                          ANTIC
3 SDMCTL=559:REM          copia de DMACTL
4 GRACTL=53277:REM        Registro de control gráfico
                          GTIA/CTIA
5 HPOSP0=53248:REM        Registro de posición horizon-
                          tal de I0(0)
6 PCOLR0=704:REM          Copia del registro de color de
                          I0(P0)
7 SIZEP0=53256:REM        registro de control de ancho del
                          jugador
8 GPRIOR=623:REM          registro de control de prioridad
10 GRAPHICS 7
```

```
20 SETCOLOR 4,8,4
30 SETCOLOR 2,0,0
40 COLOR 3
50 FOR Y=0 TO 79:REM          Este bucle llena la pantalla
60 PLOT 0,Y
70 DRAWTO 159,Y
80 NEXT Y
90 A=PEEK(RAMTOP)-20:REM      Debe dejarse más espacio para
                              GR.7
100 POKE PMBASE,A
110 MIBASEJP=256*A
120 POKE SDMCTL,46
130 POKE GRACTL,3
140 POKE HPOSP0,100
150 FOR I=MIBASEJP+512 TO MIBASEJP+640
160 POKE I,255:REM            Todo el area de jugador de un
                              solo color
170 NEXT I
180 POKE PCOLR0,88
190 POKE SIZEP0,3:REM         Jugador de ancho cuádruple
200 POKE GPRIOR,4:REM         Establecimiento de prioridad
210 COLOR 4
220 FOR Y=30 TO 40
230 PLOT Y+22,Y
240 DRAWTO Y+43,Y
250 NEXT Y
```

El programa produce el siguiente despliegue:

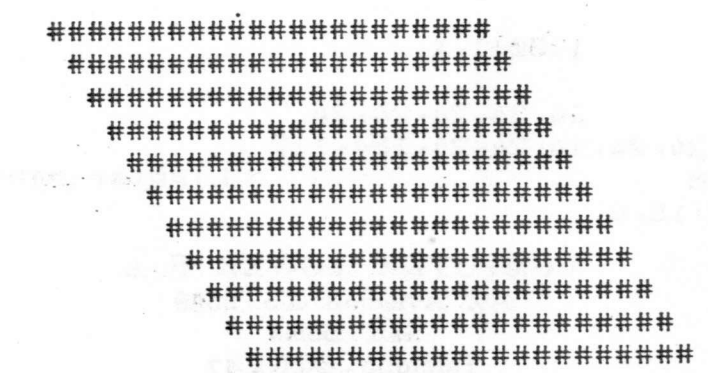


Figura 4.3
Enmascarando un jugador para mayor resolución

Otra aplicación de la gráfica de jugador-proyectil es la creación de caracteres especiales. Existen muchos tipos de caracteres que exceden los límites verticales de un juego de caracteres normales. Una forma de enfrentar esto es creando un juego de caracteres especiales para estos casos. Otra forma es usar jugadores. Subscriptores, signos de integración y otros símbolos especiales pueden realizarse por esta vía. Un ejemplo de programa para ello es:

```
1 RAMTOP=106:REM                              Puntero comienzo de RAM
2 PMBASE=54279:REM                            Puntero ANTIC a RAM J/P
3 SDMCTL=559:REM                              Copia de DMACTL
4 GRACTL=53277:REM                            Registro de control
                                              gráfico CTIA/GTIA
5 HPOS0=53248:REM                             Registro de posición
                                              horizontal de I0(P0)
6 PCOLR0=704:REM                              Copia del registro de
                                              color I0(P0)
10 GRAPHICS 0:A=PEEK(RAMTOP)-16:REM Espacio necesario para
                                              resolución de una línea
20 POKE PMBASE,A
30 MIBASEJM=256*A
40 POKE SDMCTL,62
50 POKE GRACTL,3
60 POKE HPOS0,102
70 FOR I=MIBASEJM TO MIBASEJM+1280
80 POKE I,0
90 NEXT I
100 POKE PCOLR0,154
110 FOR I=0 TO 15
120 READ X
130 POKE MIBASEJM+1100+I,X
140 NEXT I
150 DATA 14,29,24,24,24,24,24,24
160 DATA 24,24,24,24,24,24,184,112
170 ? " ":REM                                 Limpiar pantalla
180 POSITION 15,6
190 ? "xdx"
```

Este programa produce el siguiente despliegue:

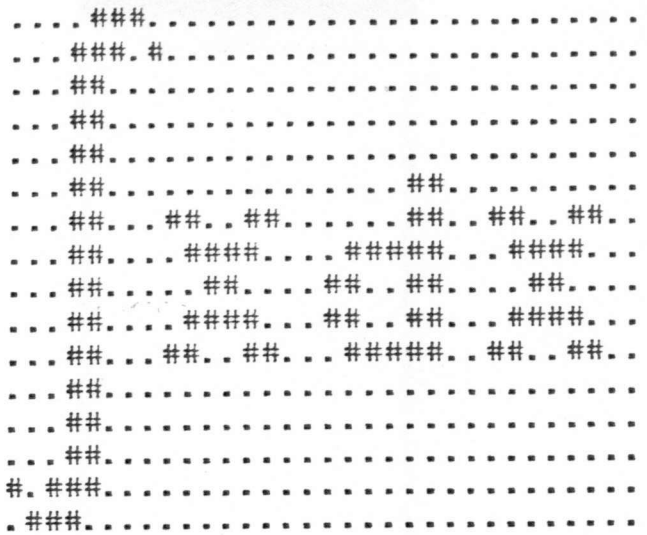


Figura 4.4
Usando un jugador como carácter especial

Una aplicación particularmente útil para los jugadores es la de los cursores. Con su habilidad de moverse suavemente a cualquier lugar de la pantalla sin alterar el contenido de ésta, se prestan idealmente para aplicaciones de este tipo. El cursor puede cambiar de color a medida que se mueva sobre la pantalla para indicar que es lo que tiene bajo sí.

La gráfica de jugador-proyectil provee muchas capacidades. Su uso para juegos de acción como objetos animados son obvios. Tiene también usos mucho más serios. Puede agregar color y resolución a cualquier despliegue. Puede representar caracteres especiales. Puede emplearse para cursores. Usela.

CAPITULO 5
INTERRUPCIONES DE LA LISTA DE DESPLIEGUE

La interrupción de lista de despliegue es una de las capacidades más poderosas comprendidas en el Sistema Computacional Personal ATARI. También constituye una de las herramientas menos accesibles del sistema, ya que requiere una clara comprensión de lenguaje ensamblador, lo mismo que de todas las demás características de la máquina. Las interrupciones de lista de despliegue por sí solas no dan capacidades adicionales; deben usarse en conjunto con otros potenciales del sistema, tales como la gráfica de jugador-proyectil, la indirección de juegos de caracteres o la indirección de registros de color. Con interrupciones de lista de despliegue puede mostrarse todo el potencial de estas características.

Las interrupciones de lista de despliegue sacan provecho de la característica secuencial del despliegue de televisión por exploración de barrido. La televisión traza la imagen de pantalla en forma secuencial. Traza las imágenes desde la parte más alta de la pantalla hasta su parte más baja. Este trazado demora alrededor de 13.000 microsegundos, lo que parece instantáneo al ojo humano, pero constituye un tiempo bastante largo en la escala en que trabaja el computador. Este tiene suficiente tiempo para cambiar los parámetros del despliegue de pantalla mientras lo traza. Naturalmente, debe ejecutar cada cambio cada vez que se dibuja la pantalla, lo que sucede 60 veces por segundo. También, (y aquí está la parte complicada), debe efectuar el cambio del parámetro en cuestión en exactamente el mismo instante cada vez que se traza la pantalla. Es decir, el ciclo de cambio de parámetros de pantalla debe sincronizarse con el ciclo de trazado de pantalla. Una forma de hacerlo podría ser enclavando el 6502 en un bucle de tiempo muy preciso, cuya frecuencia de ejecución sea exactamente 60 Hertz. Esto dificultaría de sobremanera realizar cálculos que fueran distintos a los del despliegue de pantalla. También sería una labor muy tediosa. Una alternativa mejor sería interrumpir al 6502 justamente antes que llegue el tiempo de cambiar los parámetros de pantalla. El 6502 responde a esta interrupción, cambia los parámetros de pantalla y retorna a su ocupación normal. La interrupción debe producirse exactamente en el mismo instante cada vez que que se realiza el trazado de pantalla. Esta interrupción sincronizada especialmente, la provee ANTIC; y se le llama interrupción de lista de despliegue: ILD (en inglés: 'Display List Interrupt': DLI).

El ajuste de tiempo y la ejecución de cualquier proceso

de interrupción suele ser intrincado; por ello primero se describirá la secuencia de hechos en una interrupción de lista de despliegue que opera correctamente. El proceso comienza cuando ANTIC encuentra una instrucción de lista de despliegue con su bit de interrupción (el bit D7) puesto. ANTIC espera hasta que se haya trazado la última línea de barrido de la línea de modo en la que se encuentra en ese momento. ANTIC, a continuación se refiere a su registro NMIEN para ver si se ha habilitado una interrupción de lista de despliegue. Si el bit de habilitación no está puesto, ANTIC ignora la interrupción y continua con su labor regular. Si el bit de habilitación está puesto, ANTIC baja la linea NMI del 6502. A continuación ANTIC continua con sus labores normales de despliegue. El 6502 salta a través del vector NMI a una rutina de servicio de interrupción del Sistema Operativo. Esta rutina primero determina la causa de la interrupción. Si realmente la interrupción es de lista de despliegue, la rutina apunta a través de las direcciones $0200, $0201 (bajo, alto) a una rutina de servicio de interrupción de lista de despliegue. La rutina ILD cambia uno o más de los registros gráficos que controlan el despliegue. Enseguida el 6502 retorna de la rutina de interrupción para continuar con su programa principal.

Hay una serie de pasos envueltos en la preparación de una interrupción de lista de despliegue. Lo primero que debe hacerse es escribir la rutina de interrupción de lista de despliegue propiamente tal. La rutina debe empujar sobre el stack todo registro del 6502 que se pueda alterar, ya que la rutina de supervisión de interrupción del Sistema Operativo no guarda los registros (El 6502 si empuja automaticamente el registro de estado del procesador al stack). La rutina debe ser corta y rápida y cambiar solamente los registros relacionados con el despliegue. Deberìa terminar reponiendo los registros del 6502 que se hubieran llevado al stack. A continuación la rutina de servicio de interrupción de lista de despliegue debe ubicarse en alguna parte de la memoria. La página 6 constituye un lugar ideal. Ponga los vectores de las direcciones $0200, $0201 para que apunten a su rutina. Determine el punto vertical de la pantalla en el que Ud. quiera que ocurra la interrupción de lista de despliegue, enseguida ubique la instrucción de lista de despliegue correspondiente y ponga el bit D7 de la instrucción inmediatamente anterior. Por último habilite la interrupción de lista de despliegue poniendo el bit D7 del registro NMIEN en la dirección $D40E. La interrupción de lista de despliegue comenzará a operar de inmediato.

Al igual que en cualquier rutina de servicio de interrupción, las consideraciones de tiempo pueden ser crìticas. ANTIC no envìa la interrupción al 6502 de inmediato apenas encuentra la instrucción; retarda la instrucción hasta

haber terminado con el barrido de la última línea de la línea de modo en que se efectúa la interrupción. El 6502 y la rutina de servicio de interrupción del Sistema Operativo en conjunto consumen entre 18 y 25 ciclos de máquina. Así, la primera instrucción de su rutina de servicio de interrupción de lista de despliegue no será alcanzada por lo menos hasta 18 ciclos de máquina después de la última línea de barrido de la línea de modo que interrumpe. 18 ciclos de máquina corresponden a 36 compases de color de la pantalla. Así, la rutina de servicio de interrupción de lista de despliegue comenzará a ejecutarse mientras el haz de electrones va cruzando la pantalla en la última línea de barrido de la línea de modo que interrumpe. Por ejemplo, si esta rutina de interrupción de lista de despligue cambia un registro de color, el color antiguo se desplegará en la mitad izquierda de la pantalla y el color nuevo aparecerá sobre el lado derecho. A raíz de las imprecisiones de tiempo en respuesta del 6502 a una interrupción, el borde entre los dos no será preciso sino que se moverá hacia atrás y hacia adelante en forma irritante.

Hay sin embargo una solución para este problema. Se da en la forma del registro WSYNC (espera de sincronismo horizontal). Cada vez que bajo cualquier forma se direcciona este registro, ANTIC baja la línea RDY del 6502. Esto realmente congela al 6502 hasta que el registro sea puesto por un sincronismo horizontal. El resultado es que el 6502 se mantiene estático hasta que el haz de electrones vuelva al borde izquierdo de la pantalla. Si se inserta una instrucción STA WSYNC justamente antes de la instrucción que almacena un valor en el registro de color, el color ingresará mientras el haz está fuera del borde izquierdo de la pantalla. La transición de color ocurrirá una línea de barrido más abajo, pero será limpia y definida.

Por lo tanto, la forma correcta de usar una interrupción de lista de despliegue, es poner el bit de interrupción de lista de despliegue en la línea de modo anterior a la línea de modo en la cual queramos que ocurra la acción. La rutina de servicio de interrupción de lista de despliegue primero debe guardar los registros del 6502 en el stack, enseguida cargar los registros del 6502 con los nuevos valores gráficos que quieran usarse. Debe ejecutar una STA WSYNC, y enseguida poner los nuevos valores en los registros apropiados de ANTIC o GTIA. Por último, debe restaurar los registros del 6502 y retornar de la interrupción. Este procedimiento garantizará que los registros gráficos se cambien al comienzo de la línea deseada y mientras el haz de electrones se encuentre fuera de la pantalla.

A continuación se da un programa demostrativo simple de una interrupción de lista de despliegue:

```
10 LISDES=PEEK(560)+256*PEEK(561):REM  ubicar lista de
                                       despliegue
20 POKE LISDES+15,130:REM              insertar instrucción
                                       de interrupción
30 FOR I=0 TO 19:REM                   bucle para ingresar
                                       rutina servicio ILD
40 READ A:POKE 1536+I,A:NEXT I
50 DATA 72,138,72,169,80,162,88
60 DATA 141,10,212,141,23,208
70 DATA 142,24,208,104,170,104,64
80 POKE 512,0:POKE 513,6:REM           vector de la
                                       interrupción
90 POKE 54286,192:REM                  habilitar ILD
```

Esta rutina usa la siguiente rutina de servicio de interrupción de lista de despliegue en lenguaje assembler:

```
PHA             guardar acumulador
TXA
PHA             guardar registro X
LDA #$50        color oscuro para caracteres
LDX #$58        rosado
STA WSYNC       espera
STA COLPF1      almacenar color
STX COLPF2      almacenar color
PLA
TAX
PLA             restaurar registros
RTI             listo
```

Es esta una rutina de interrupción de lista de despliegue muy simple. Cambia el color de fondo de azul a rosado. También cambia el color de los caracteres de modo que ellos aparezcan obscuros contra el fondo rosado. Uno se pregunta por qué la parte superior de la pantalla permanece azul a pesar de que la rutina de interrupción de lista de despliegue permanentemente está colocando el color rosado en el registro. La respuesta es que la rutina de interrupción de borrado vertical del Sistema Operativo también sigue colocando azul en el registro de color durante el periodo del borrado vertical. El color azul proviene del registro de copia del Sistema Operativo para ese registro de color. Cada registro de color de circuito tiene su copia en RAM. Puede que ya conozca estos registros de copia en las ubicaciones 708 hasta 712. Para la mayoría de los propósitos es suficiente para cambiar color hacer un POKE de los valores correspondientes en los registros de copia. Si se hace el POKE directamente a los registros de circuito, el proceso de copia del Sistema

Operativo los borrará en el lapso de un 1/60 de segundo; sin embargo, para interrupciones de lista de despliegue debe almacenarse los valores del color nuevo directamente el los registros de circuito. No se puede usar una interrupción de lista de despliegue para fijar el color de la primera línea desplegada en la pantalla. El Sistema Operativo se hace cargo de esto. Use las interrupciones de lista de despliegue para cambiar colores de las líneas que sigan a la primera.

Cargando los colores directamente en los registros de circuito se crea un problema nuevo: se inhibe el modo automático de rotación de colores. El modo de rotación de colores es una característica provista por el Sistema Operativo. Después de unos nueve minutos sin presionar ninguna tecla, los colores de la pantalla comienzan a ciclar a través de valores aleatorios a intensidad reducida. Esto asegura que un computador que se quede sin atención por algunas horas no vaya a quemar una imagen en la pantalla del televisor. Es fácil incluir el modo de rotación de colores en una interrupción de lista de despliegue. Solamente debe insertarse dos líneas del código assembler dentro de esta rutina:

```
ANTIGUO          NUEVO

LDA NEWCOL       LDA NEWCOL
STA WSYNC        EOR COLRSH
STA COLPF2       AND DRKMSK
                 STA WSYNC
                 STA COLPF2
```

DRKMSK y COLRSH son ubicaciones de página cero ($4E y $4F) establecidas y puestas al día por el Sistema Operativo durante la interrupción del borrado vertical. Cuando el modo de rotación de colores no está en efecto, COLRSH toma el valor 0 y DRKMSK toma $FF. Cuando el modo de rotación está actuando, COLRSH toma un nuevo valor aleatorio cada 4 segundos y DRKMSK mantiene el valor $F6. Así, COLRSH revuelve los colores y DRKMSK corta el bit más alto de la luminancia.

La implementación del modo de rotación de colores en una interrupción de lista de despliegue agudiza un problema ya difícil: la falta de tiempo de ejecución durante el intervalo de lista de despliegue. Una descripción de los tiempos durante una interrupción de lista de despliegue hará más obvio el problema. La ejecución de una interrupción de lista de despliegue se puede desglosar en tres fases. La fase uno cubre el período desde el comienzo de la interrupción de lista de despliegue hasta la instrucción STA WSYNC . Durante la fase uno el haz de electrones está trazando la primera línea de barrido de la línea de modo que interrumpe. La fase dos cubre

el periodo desde la instrucción STA WSYNC hasta la aparición del haz sobre la pantalla del televisor. La fase dos corresponde al borrado horizontal; todos los cambios gráficos deberian efectuarse durante esta fase. La tres cubre el periodo desde la aparición del haz sobre la pantalla hasta el final de la rutina de servicio de la interrupción de lista de despliegue. Los tiempos de la fase tres no son criticos.

Cada línea de barrido horizontal requiere 114 compases de reloj de tiempo real. La interrupción de lista de despliegue llega al 6502 alrededor del compás número 15. El 6502 demora alrededor de 7 periodos para responder a la interrupción. La rutina del Sistema Operativo que sirve la interrupción y la apunta hacia la rutina de servicio de interrupción de lista de despliegue, requiere 11 ciclos de máquina. Asi, la rutina de servicio de la interrupción de lista de despliegue no se alcanza hasta unos 33 compases de reloj después. Además, la instrucción STA WSYNC debe haber comenzado en el ciclo número 103; esto reduce el tiempo disponible para la fase 1 en 11 ciclos.
Por último, el acceso directo de memoria de ANTIC requerirá algunos de los ciclos de reloj remanentes. Se pierden 9 ciclos en el refrescamiento de memoria del acceso directo de memoria. Esto deja un máximo absoluto de 61 ciclos disponibles para la fase 1. Este máximo se alcanza solamente con lineas de modo en blanco. Las instrucciones de mapa y de carácter darán por resultado la pérdida de un ciclo adicional por cada byte de despliegue. Las peores condiciones se dan con los modos BASIC 0, 7 y 8, que requieren 40 bytes por linea. En estos modos solamente 21 ciclos de máquina están disponibles para la fase uno. Asi, una rutina de fase uno tendrá entre 21 a 61 ciclos de máquina disponibles como tiempo de ejecución.

La fase dos, la fase crítica, se extiende por los 24 ciclos de reloj de tiempo real. Al igual que en la fase uno, algunos de estos ciclos se pierden en el acceso directo de memoria. La gráfica de jugadores-proyectiles quitará unos cinco ciclos si se está usando. Las instrucciones de despliegue requieren un ciclo; si se usa la instrucción de carga de barrido de memoria se necesitan dos ciclos más. Por último, uno o dos ciclos se pierden debido a refrescamiento de memoria o reubicación de datos de despliegue. Asi, solamente de 14 a 23 ciclos de máquina están disponibles para la fase dos.

Asi llega a hacerse obvio el problema de tiempo en las interrupciones de lista de despliegue. Para cargar, rotar y almacenar un solo color, se consumen alrededor de 14 ciclos. Poner el acumulador y los registros X e Y sobre el stack y enseguida recargando, rotando y almacenando 3 colores en el acumulador el registro X y el registro Y, costará 47 ciclos,

es decir la mayor parte si no todo el tiempo disponible de la fase uno. Obviamente el programador que quiere usar las interrupciones de lista de despliegue para cambios gráficos importantes, destinará mucho esfuerzo a los tiempos de la interrupción. Afortunadamente, el programador novicio no necesita preocuparse demasiado con cálculos de tiempo muy extensos. Si sólo se quiere hacer un cambio de color o alguna operación gráfica simple, se hace innecesaria la optimización de velocidad y el conteo de ciclos. Son éstas consideraciones, que solamente son importantes para situaciones muy complejas.

No existe opción simple para el programador que tiene que cambiar más de tres registros de color en una sola interrupción de lista de despliegue. Tal vez sea posible cargar, rotar y almacenar un cuarto color en los comienzos de la fase tres si ese color no se despliega en la parte izquierda de la pantalla. Igualmente, un color que no aparezca sobre el lado derecho de la pantalla, podría cambiarse durante la fase uno. Otra forma es descomponer una interrupción de lista de despliegue demasiado activa en dos menos ambiciosas, cada una de las cuales hace la mitad del trabajo de la original. La segunda interrupción de lista de despliegue podría darse probablemente, insertando una sola línea de barrido con una instrucción en blanco (naturalmente con el bit de interrupción de lista de despliegue puesto) en la lista de despliegue justamente debajo de la linea de modo de interrupción principal. Naturalmente esto requiere de algún espacio de pantalla.

Otra solución parcial es realizar los trabajos de rotación durante los perìodos de borrado vertical. Para hacerlo, deben mantenerse dos tablas en RAM. La primera tabla contiene los valores de color que se esperan desplegar por las rutinas de interrupcion de lista de despliegue. La segunda tabla contiene los valores de rotación de estos colores. Durante el borrado vertical una rutina de servicio de interrupción realizada por el programador recoge cada color de la primera tabla, lo rota y lo almacena en su valor rotado en la segunda tabla. La rutina de interrupción de lista de despliegue a continuación reencuentra los valores directamente de la segunda tabla sin requerirse tiempo adicional por esta rotación.

Muchas veces se desea que una serie de interrupciones de lista de despliegue ocurra en diferentes posiciones verticales sobre la pantalla. Es éste un camino importante para agregar más color al despliegue. Desafortunadamente hay un solo vector de interrupción de lista de despliegue. Si se quiere implementar varias interrupciones, entonces la vectorización hacia la interrupción que corresponda debe ser implementada por la misma rutina de interrupción de lista de

despliegue. Hay varias formas de lograrlo. Si la rutina de interrupción ejecuta el mismo proceso solamente con diferentes valores, entonces puede recurrirse a una tabla. En cada paso a través de la rutina, se incrementa un contador y se le usa como índice a la tabla de valores. Una rutina ejemplo de interrupción de lista de despliegue que hace esto sería como lo que sigue:

```
        PHA
        TXA
        PHA
        INC CONTAD
        LDX CONTAD
        LDA TABCOL,X    use la pág. $F0 para la tabla de co-
                        lores
        STA WSYNC       espera de sincronismo horizontal
        STA COLBAC
        CPX #$4F        última línea ?
        BNE FINILD      no, salida
        LDA #$00        si, reponer contador
        STA CONTAD
FINILD  PLA
        TAX
        PLA             restaurar acumulador
        RTI
```

El siguiente es un programa BASIC para llamar esta rutina:

```
10  GRAPHICS 7
20  LISDES=PEEK(560)+256*PEEK(561):REM   encontrar lista de
                                          despliegue
30  FOR J=6 TO 84:REM                     dar ILD a cada línea
40  POKE LISDES+J,141:REM                 línea de modo BASIC 7
                                          con bit de ILD pues-
                                          to
50  NEXT J
60  FOR J=0 TO 30
70  READ A:POKE 1536+J,A:NEXTJ:REM        poner en RAM la ru-
                                          tina de servicio ILD
80  DATA 72,138,72,238,32,6,175,32,6
90  DATA 189,0,240,141,10,212,141,26,208
100 DATA 224,79,208,5,169,0
110 DATA 141,32,6,104,170,104,64
120 POKE 512,0:POKE 513,6:REM             vector a la rutina
                                          de servicio de ILD
130 POKE 54286,192:REM                    habilitar ILD
```

Este programa pondrá 80 colores diferentes sobre la pantalla.

Hay otras formas para implementar interrupciones de lista de despliegue múltiples. Una forma es usar un contador de ILD para apuntar a la rutina de servicio ILD que corresponda en cada caso. Esto reduce la velocidad de respuesta de todas las interrupciones de lista de despliegue, particularmente aquellas que se encuentran al final de la secuencia. Otra forma es hacer que cada rutina de servicio de interrupción de lista de despliegue escriba la dirección de la rutina siguiente en la ubicacion del vector de interrupción de lista de despliegue en $200 y $201. Esto deberia hacerse durante la fase tres. Es la solución más general al problema de las interrupciones de lista de despliegue múltiples. Tiene la ventaja adicional que la lógica de los vectores se realiza después de la parte crìtica del tiempo de la interrupción de lista de despliegue y no antes.

La rutina de teclado del Sistema Operativo interfiere con la operación de una interrupción de lista de despliegue. Cada vez que se reconoce la presión sobre una tecla, el parlante emite un sonido. El tiempo preciso para este sonido lo dan algunas instrucciones STA WSYNC.

Esto puede desequilibrar los tiempos de una rutina de interrupción de lista de despliegue y hacer que los colores de la pantalla salten hacia abajo en una lìnea de barrido por una fracción de segundo. No existe una solución simple para este problema. Una posibilidad contempla el registro VCOUNT, un registro de lectura solamente que se encuentra en ANTIC y que indica qué lìnea está desplegando ANTIC en ese momento. Una rutina de interrupción podrìa examinar este registro para decidir cuando debe cambiarse un color. Otra solución consiste en inhibir la rutina de servicio de teclado del Sistema Operativo y proveer una rutina de teclado propia. Esto serìa una labor bastante tediosa. La última solución es no aceptar ingresos desde el teclado. Si no se reconocen las presiones sobre el teclado, no se producirá tampoco el salto en la pantalla.

La interrupción de lista de despliegue se diseñó para reemplazar una técnica de software y circuito más primitiva llamada nuclear. Un núcleo es un bucle de programa de 6502 que está muy bien ajustado en tiempo al ciclo de despliegue del televisor. Monitoreando el registro VCOUNT y consultando una tabla de cambios de pantalla catalogados en función de los valores de VCOUNT, el 6502 puede controlar arbitrariamente todos los valores gráficos de toda la pantalla. Se paga un valor muy alto por esta posibilidad. El 6502 no queda dispobible para cálculos durante todo el tiempo de despliegue

de pantalla que corresponde aproximadamente a un 75% del tiempo total. Además, ningún cálculo puede consumir más de 4000 ciclos de máquina aproximadamente, que son los disponibles durante el intervalo de borrado vertical y periodo de sobrebarrido. Esta restricción indica que los núcleos solamente pueden usarse con programas que requieran pocos cálculos, como por ejemplo algunos programas de destreza y acción. Por ejemplo, el programa BASKETBALL para ATARI 400/800 usa un núcleo; el programa requiere pocos cálculos pero mucho color. Los jugadores multicolores de este juego no podrian realizarse con interrupciones de lista de despliegue, porque las interrupciones de lista de despliegue están referidas a las posiciones verticales del campo, no a las posiciones de los jugadores.

Es posible extender la idea del núcleo hasta una sola linea de barrido y realizar cambios en los registros gráficos durante su transcurso. Asi un mismo registro de color puede presentar varios colores sobre una misma linea de barrido. La posición horizontal del cambio de color se determina por el tiempo que transcurre antes que se produzca el cambio. Asi, contando cuidadosamente los ciclos de máquina, el programador puede lograr más gráfica sobre la pantalla. Desafortunadamente es muy difícil conseguirlo en la práctica. Con una ANTIC accediendo directamente a la memoria del 6502 es muy difícil saber exactamente cuantos ciclos han transcurrido realmente. No es suficiente un simple conteo de los ciclos del 6502. Si se desconecta el acceso direto de memoria de ANTIC, el 6502 puede asumir todo el control del despliegue, pero debe en ese caso realizar también todo el trabajo que normalmente efectúa ANTIC. Por estas razones raras veces vale la pena recurrir a un núcleo horizontal. Sin embargo, si las dos imágenes que deben desplegarse con colores diferentes están muy separadas, digamos unos 20 compases de color o más, la separación deberia compensar la inseguridad de tiempo y hacer posible esta técnica.

Se hace obvio ahora el enorme valor de la indirección gráfica y de todos estos registros modificados. Con interrupciones de lista de despliegue cada uno de estos registros puede cambiarse al vuelo. Puede ponerse mucho color, mucha gráfica y efectos especiales sobre la pantalla. La aplicación mas obvia de una interrupción de lista de despliegue es poner más color. Cada registro de color se puede cambiar tantas veces como interrupciones de lista de despliegue haya disponibles. Esto se aplica tanto a los registros de color de los campos como de los jugadores. En esta forma, hay hasta nueve registros de color cada uno de los cuales puede desplegar hasta 128 colores diferentes. No es esto suficiente? Naturalmente, un programa normal no se prestará por si sólo para usar con eficacia todos estos colores. A la larga demasiadas interrupciones de lista de despliegue

terminan por disminuir la velocidad de todo el programa. A veces un determinado despliegue no puede acomodar muchas interrupciones. En la práctica, una docena de colores es fácil, dos docenas requieren una planificación cuidadosa y más de esto requiere en general situaciones muy conflictivas.

Las ILD pueden dar mucho color; también, pueden usarse para extender el potencial de las gráficas de jugadores-proyectiles. La posición horizontal de un jugador puede cambiarse por una interrupción. Asi un jugador puede reusarse más abajo en la pantalla. Un solo jugador puede tener varias encarnaciones en la pantalla. Si nos imaginamos al jugador como una columna vertical con imágenes sobre ella, una interrupción de lista de despliegue se convierte en una tijera con la cual se puede recortar la columna y reubicar sus secciones en la pantalla. Naturalmente nunca dos secciones del jugador pueden encontrarse sobre la misma línea horizontal, así las dos encarnaciones del jugador no pueden encontrarse a la misma altura. Si sus requerimientos de despliegue requieren objetos gráficos que nunca se encontrarán sobre la misma línea horizontal, un solo jugador puede resolver el problema.

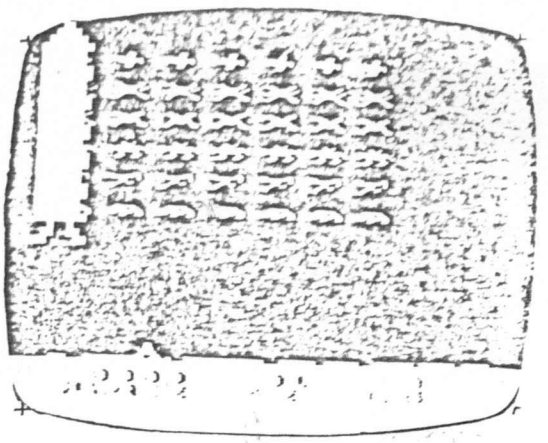
Otra forma en la que se pueden usar las interrupciones de lista de despliegue en conjunto con jugadores, es el cambio de ancho o prioridad. Esto probablemente se use con frecuencia con el truco de plantilla por prioridad descrito en el Capìtulo 4.

La última aplicación de las ILD es el cambio de juego de caracteres a media altura de la pantalla. Esto permite que un programa use gráfica de caracteres en una ventana mayor y texto normal en una ventana de texto. Se posibilitan también cambios múltiples de juegos de caracteres; un programa podrìa usar un juego de caracteres en la parte alta de la pantalla, otro juego de caracteres gráficos al centro de la pantalla y caracteres de texto normal en la parte baja. También podrìa tenerse diferentes tipografias de texto sobre la misma pantalla. El bit de reflexión vertical podrìa cambiarse con una rutina de interrupción de lista de despliegue haciendo que parte del texto apareciera de cabezas y otra parte del texto al derecho.

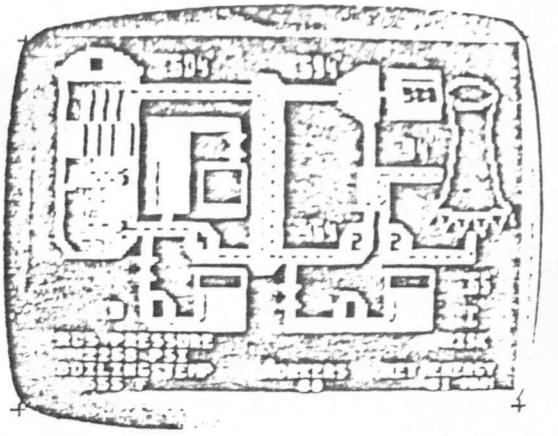
El buen uso de las interrupciones de lista de despliegue requiere de un acabado estudio del despliegue de pantalla. El diseñador debe dar cuidadosa consideración a la arquitectura vertical del despliegue. El sistema de televisión por barrido no es simétrico en ambas direcciones; tiene mucho mayor estructura vertical que horizontal. Ello se debe al ritmo de trazado que es unas 200 veces mas rápido en el sentido

horizontal que en el vertical. El sistema de despliegue ATARI se diseñó específicamente para televisión por barrido y refleja la anisotropia de los sistemas de barrido. El despliegue ATARI no es una hoja de papel en blanco en la cual Ud. puede trazar, es una pila de tiras muy angostas cada una de las cuales puede tomar diferentes parámetros. El programador que insista diseñando un despliegue isotrópico pierde muchas oportunidades. En cambio, obtendrá óptimos resultados al organizar la información que quiera desplegar con una fuerte estructura vertical. Esto permite aprovechar al máximo el potencial de las interrupciones de lista de despliegue.

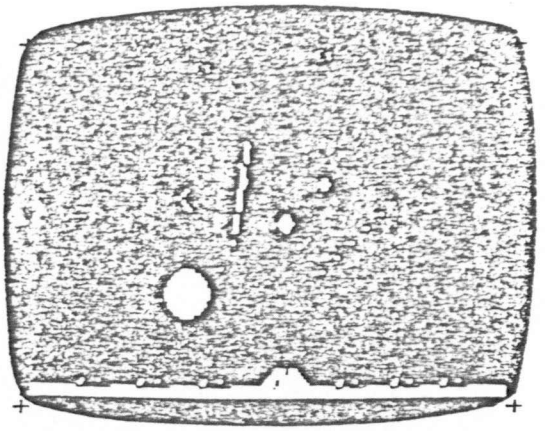
La Figura 5.1 muestra los despliegues de algunos programas y da estimaciones del grado de arquitectura vertical usado en cada uno de ellos.



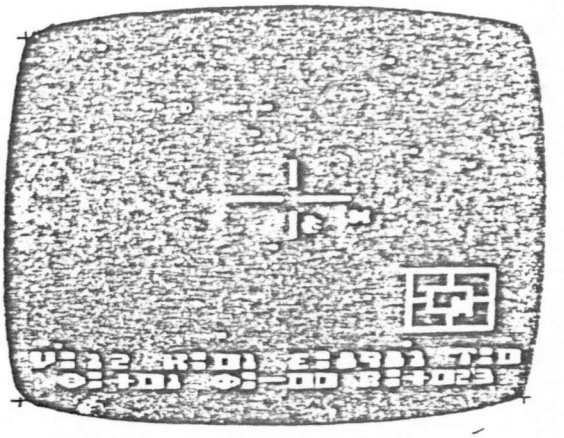
SPACE INVADERS
(M. reg. de Taito America Corp.)
MUCHA



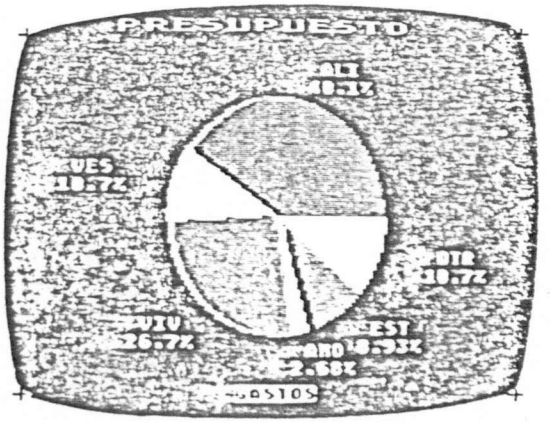
SCRAM
(Simulación reactor nuclear)
POCA



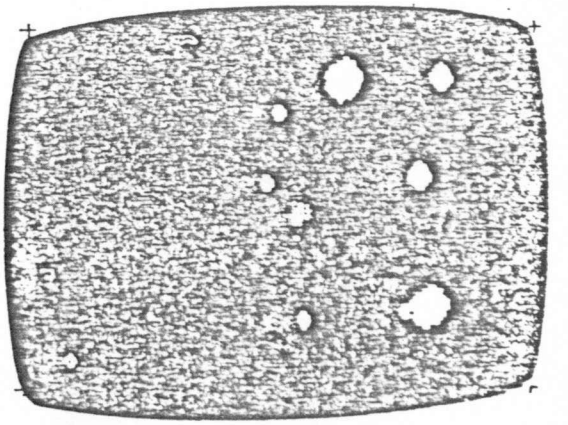
MISSILE COMMAND
ALGUNA



STAR RAIDERS
POCA



GRAFIQUELO
NINGUNA



ASTEROIDS
NINGUNA

Figura 5.1
ejemplos de arquitectura vertical de pantalla

CAPITULO E
MOVIMIENTO DE PANTALLA

Frecuentemente la información que el programador desea desplegar excede a la cantidad que puede hacerse caber en la pantalla. Una forma de resolver el problema es mover la información a través del despliegue. Por ejemplo, los listados de programas BASIC se mueven verticalmente desde la parte de abajo hasta la parte de arriba de la pantalla. Todos los computadores personales poseen este tipo de movimiento. Sin embargo, el sistema de Computación Personal ATARI tiene dos facilidades de movimiento adicionales, que ofrecen posibilidades excitantes. La primera es el movimiento grueso de la carga de exploración de memoria; la segunda es el movimiento fino.

Los computadores convencionales usan el movimiento grueso; en este tipo de movimiento los pixels que contienen a los caracteres se fijan en posición sobre la pantalla y el texto se desplaza moviendo los bytes a través de la RAM de pantalla. La resolución del movimiento es un pixel de carácter, lo cual es muy grueso. El movimiento producido asi es a saltos y bastante desagradable. Además se logra moviendo miles de bytes a través de la memoria, lo cual constituye una tarea lenta y burda. En esencia, el programa debe mover los datos a través del campo para producir el movimiento.

Algunos computadores personales pueden lograr un movimiento algo más fino, trazando imágenes en un modo gráfico de mayor resolución y moviendo esas imágenes. A pesar de que se consigue una mayor resolución de movimiento, deben desplazarse más datos para lograr el movimiento y por lo tanto el programa se vuelve más lento. El problema fundamental es que el movimiento se implementa, desplazando los datos a través del área de pantalla.

Existe una mejor manera de lograr movimientos gruesos con los computadores ATARI: mover el área de pantalla por sobre los datos. Los códigos de operación de la lista de despliegue soportan la llamada Carga de Exploración de Memoria (CEM o LMS). Esta instrucción se describió inicialmente en el Capítulo 2. La instrucción de carga de exploración de memoria indica a ANTIC donde se encuentra la memoria de pantalla. Una lista de despliegue normal tendrá una sola instrucción de carga de exploración de memoria, al comienzo de la lista de despliegue; el área de RAM a la cual apunta, contiene los datos para toda la pantalla en una secuencia lineal. Manipulando los bytes de operando de la instrucción de carga

de exploración de memoria, puede implementarse un movimiento primitivo. En efecto, se mueve la ventana del campo a través de los datos de pantalla. Así, manipulando solamente dos bytes de direccionamiento, puede producirse un efecto idéntico al de mover toda la memoria de pantalla. El siguiente programa justamente hace esto:

```
10 LDES=PEEK(560)+256*PEEK(561):REM  encontrar lista de des-
                                     pliegue
20 CEML=LDES+4:REM                   byte menos significati-
                                     vo del operando de CEM
30 CEMH=LDES+5:REM                   byte más significativo
                                     del operando de CEM
40 FOR I=0 TO 255:REM                bucle exterior
50 POKE CEMH,I
60 FOR J=0 TO 255:REM                bucle interior
70 POKE CEML,J
80 FOR T=1 TO 50:NEXT T:REM          bucle de retardo
90 NEXT J
100 NEXT I
```

Este programa barre con el despliegue por sobre todo el espacio de direccionamiento del computador. Todo el contenido de la memoria se lleva a la pantalla. El movimiento resultante es un movimiento serial burdo que combina movimientos horizontal y vertical. Un movimiento vertical puro puede lograrse agregando o restando una cantidad fija (el largo de linea expresado en bytes) al operando de la carga de exploración de memoria. El siguiente programa lo hace:

```
10 GRAPHICS 0
20 LDES=PEEK(560)+256*PEEK(561)
30 CEML=LDES+4
40 CEMH=LDES+5
50 PANTALLAL=0
60 PANTALLAH=0
70 PANTALLAL=PANTALLAL+40:REM              Línea siguiente
80 IF PANTALLAL<256 THEN GOTO 120:REM      excedido?
90 PANTALLAL=PANTALLAL-256:REM             sí, ajustar pun-
                                           teros
100 PANTALLAH=PANTALLAH+1:REM              de pantalla
110 IF PANTALLAH=256 THEN END
120 POKE CEML,PANTALLAL
130 POKE CEMH,PANTALLAH
140 SOUND 0,PANTALLAL,10,10:REM            lo que no se ve,
150 SOUND 1,PANTALLAH,10,10:REM            se oye
160 FOR TIEMPO=1 TO 20:NEXT TIEMPO
170 GOTO 70
```

Un movimiento puramente horizontal no es tan fácil de

conseguir como uno puramente vertical. El problema está en que la memoria RAM de pantalla para una lista de despliegue simple está organizada en forma serial. Los bytes de información de pantalla para líneas sucesivas están en secuencia con los bytes de una línea siguiendo inmediatamente a los de la línea precedente. Podemos mover horizontalmente las líneas, desplazando todos los bytes hacia la izquierda; esto se logra disminuyendo el operando de la carga de exploración de memoria. Sin embargo, el byte de más a la izquierda de cada línea se moverá hacia la posición de más a la derecha de la línea inmediatamente superior. En el primer programa ejemplo se veía este problema.

La solución está en expandir el área de datos de pantalla y partirla en una serie de áreas de datos de línea horizontal independientes. La Figura E.1 ilustra esquemáticamente esta idea:

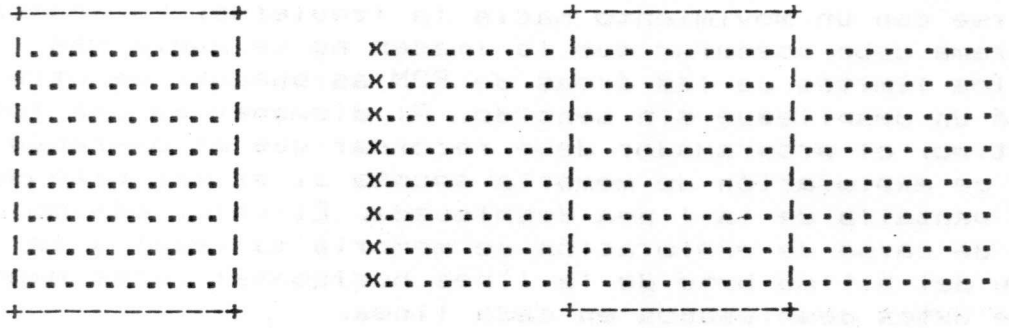


Figura E.1
ordenando la RAM de pantalla

A la izquierda tenemos el ordenamiento normal. La RAM uni-dimensional serial está apilada en secuencia lineal para crear el área de datos de pantalla. A la derecha tenemos el ordenamiento que necesitamos para un buen movimiento horizontal. Naturalmente la RAM sigue siendo unidimensional y serial, pero ahora se usa en forma diferente. La RAM para cada línea horizontal se extiende mucho más allá de lo que la pantalla puede mostrar. Esto no es un accidente, la razón del movimiento justamente es permitir al programa mostrar más información de la que se puede poner en una pantalla. No podremos mostrar toda esta información adicional si no asignamos la RAM correspondiente para mantenerla. Con este arreglo podemos implementar un verdadero movimiento horizontal. Podemos mover la ventana de pantalla por sobre los datos de pantalla sin el movimiento vertical desagradable del método anterior.

El primer paso en la implementación de un movimiento puramente horizontal es la determinación del largo total de las lineas horizontales y en consecuencia asignar la RAM correspondiente. A continuación, debemos escribir una lista de despliegue totalmente nueva con una instrucción de carga de exploración de memoria nueva en cada línea de modo. Naturalmente la lista de despliegue resultará más larga de lo normal, pero no hay ninguna razón para no escribir una lista de despliegue de este tipo. Qué valores usaremos para los operandos de carga de exploración de memoria?. Lo más conveniente es usar la dirección del primer byte de cada línea horizontal de datos de pantalla, los puntos marcados con una X en el diagrama. Habrá una de estas direcciones para cada linea de modo de la pantalla. Una vez ubicada la lista de despliegue nueva, ANTIC debe apuntarse a ella y debemos escribir datos para llenar la pantalla. Para realizar el movimiento, todos y cada uno de los operandos de carga de exploración de memoria en la lista de despliegue deben incrementarse para un movimiento hacia la derecha o disminuirse con un movimiento hacia la izquierda. La lógica del programa debe asegurar que la imagen no se mueva más allá de los límites de las áreas de RAM asignados, de otra forma resultará un despliegue sin sentido. Al disponer de una lógica de este tipo, el programador debe recordar que el operando de la carga de exploración de memoria apunta al primer byte de datos de pantalla de la linea desplegada. El valor máximo del operando de carga de exploración de memoria es igual a la dirección del último byte de la línea horizontal larga menos el número de bytes desplegados en cada linea.

Como este proceso es bastante complicado, desarrollemos un ejemplo. Primero, debemos elegir el largo total de nuestra línea horizontal. Usaremos un largo de línea horizontal de 256 bytes, ya que ello simplificará el cálculo de las direcciones. Cada linea horizontal requerirá por lo tanto una página de RAM. Como usaremos el modo BASIC 2, habrá 12 lineas de modo por pantalla; así se requerirán 12 páginas o 3 Kbytes de RAM. Por simplicidad (y para garantizar que nuestra RAM de pantalla estará repleta de datos diferentes de cero), usaremos los últimos 3 Kbyte de RAM. Esta área la emplean el sistema operativo y el sistema operativo de disco, de manera que estará llena de datos interesantes. Para hacer las cosas más interesantes aún, pondremos la lista de despliegue en la página 6, de manera que podamos desplegar la lista de despliegue en la pantalla, a medida que nos movamos. Los valores iniciales de los operandos de carga de exploración de memoria resultarán así muy fáciles de calcular; los bytes de menor significación siempre serán iguales a cero y los bytes más significativos serán (en orden) 0, 1, 2, etc. El siguiente programa realiza todas estas operaciones y mueve la pantalla horizontalmente:

```
10 REM Primero, preparemos la lista de despliegue
20 POKE 1536,112:REM                    8 líneas en blanco
30 POKE 1537,112:REM                    8 líneas en blanco
40 POKE 1538,112:REM                    8 107neas en blanco
50 FOR I=1 TO 12:REM                    bucle para armar la lista
                                        de despliegue
60 POKE 1536+3*I,71:REM                 modo BASIC 2 con carga de
                                        exploración de memoria
70 POKE 1536+3*I+1,0:REM                byte menos significativo
                                        del operando a CEM
80 POKE 1536+3*I+2,I:REM                byte más significativo
                                        del operando a CEM
90 NEXT I
100 POKE 1575,65:REM                    Instrucción ANTIC de
                                        salto y espera de borrado
                                        vertical
110 POKE 1576,0:REM                     la lista de despliegue
                                        comienza en $0600
120 POKE 1577,6
130 REM indicar a ANTIC donde se encuentra la lista de
despliegue
140 POKE 560,0
150 POKE 561,6
160 REM ahora,movimiento horizontal
170 FOR I=0 TO 235:REM                  bucle a través de los
                                        bytes menos significa-
                                        tivos de la carga de ex-
                                        ploración de memoria
175 REM usamos 235--y no 255--porque el ancho de pantalla es
de 20 caracteres
180 FOR J=1 TO 12:REM
190 POKE 1536+3*J+1,I:REM               poner nuevo byte menos
                                        significativo de la carga
                                        de exploración de memoria
200 NEXT J
210 NEXT I
220 GOTO 170:REM                        bucle infinito
```

Este programa mueve los datos de derecha a izquierda. Cuando se alcanza el final de una página simplemente vuelve a comenzar. La lista de despliegue se encuentra en la sexta línea de arriba hacia abajo (está en la página seis). Aparece como una secuencia de comillas.

El siguiente paso es mezclar el movimiento horizontal y el vertical para obtener un movimiento en diagonal. El movimiento horizontal se logra agregando uno o restando uno del operando de carga de exploración de memoria. El movimiento vertical se obtiene agregando el largo de línea o restando el largo de línea del operando de carga de exploración de memoria. El movimiento en diagonal se obtiene ejecutando ambas operaciones. Hay cuatro direcciones posibles de movimiento en

diagonal. Si, por ejemplo, el largo de la línea es de 256 bytes y queremos movernos hacia abajo y hacia la derecha debemos agregar 256+(-1)=255 a cada operando de carga de exploración de memoria de la lista de despliegue. Es ésta una suma sobre dos bytes; el programa ejemplo BASIC dado más arriba evita estas dificultades de las manipulaciones de direcciones de dos bytes, pero en la mayoría de los casos, los programas no serán tan limitados. Para un movimiento en dos dimensiones realmente rápido se hará necesario el lenguaje assembler.

Puede encontrarse todo tipo de movimientos, si manipulamos en forma diferente los bytes de la carga de exploración de memoria. Las líneas podrían moverse una con respecto a la otra o saltar una por encima de otra. Naturalmente, algunas de estas cosas pueden lograrse con despliegues convencionales, pero deberían moverse más datos para lograrlo. La verdadera ventaja del movimiento por carga de exploración de memoria está en su velocidad. En vez de manejar toda una pantalla de datos, muchos miles de bytes, un programa de este tipo solamente debe manipular dos o tres o talvez unas pocas docenas de bytes.

MOVIMIENTO DE PANTALLA FINO

La segunda facilidad importante de movimiento de pantalla en los sistemas computacionales ATARI es la capacidad de movimiento fino. Consiste en mover los pixels en pasos menores que el tamaño del pixel propiamente tal. El movimiento grueso avanza en pasos iguales a la dimensión de cada pixel; el movimiento fino avanza en pasos de una línea de barrido verticalmente o de un compás de color horizontalmente hasta completar un pixel. El movimiento fino solamente puede llevarse hasta este punto; para obtener movimiento fino completo sobre grandes distancias sobre la pantalla debemos combinar movimiento fino con movimiento grueso. (A lo largo de este capitulo llamaremos pixel al carácter entero y no a los puntos pequeños que hacen el carácter).

Hay solamente dos pasos envueltos en la implementación de un movimiento fino. Primero, ponemos el bit de habilitación de movimiento fino en los bytes de instrucción de la lista de despliegue de aquellas líneas de modo para las cuales queramos movimiento fino (en la mayoría de los casos lo querremos para toda la pantalla, de modo que pondremos los bits de habilitación de movimiento fino a lo largo de todos los bytes de instrucción de la lista de despliegue). El bit D5 de la instrucción de lista de despliegue es el bit de habilitación de movimiento vertical; el bit 4 de instrucción de lista de despliegue es el bit de habilitación de movimiento horizontal.

A continuación almacenamos el valor de movimiento deseado en el registro de movimiento correspondiente. Hay dos registros de movimiento, uno para el horizontal otro para el vertical. El registro de movimiento horizontal (HSCROL) está en $D404; el registro de movimiento vertical (VSCROL) se encuentra en $D405. Para movimiento horizontal almacenamos en HSCROL la cantidad de compases de color por la cual deseamos que se produzca el avance de movimiento de línea. Para movimiento vertical almacenamos en VSCROL la cantidad de líneas de barrido que queremos que se avance en el movimiento de las líneas de modo. Estos valores de movimiento se aplicarán a cada línea para la cual el movimiento se haya habilitado.

Hay dos factores que complican las cosas al usar movimiento fino. Ambos nacen del hecho que un despliegue movido parcialmente muestra más información que un despliegue normal. Consideremos por ejemplo lo que sucede al mover horizontalmente una línea por medio carácter hacia la izquierda. Hay 40 caracteres en cada línea. La mitad del primer carácter desaparece más allá del borde izquierdo de la pantalla. El cuadragésimo carácter se mueve hacia la izquierda. Qué aparece en su lugar?. La mitad de un carácter nuevo debería moverse para tomar el lugar del carácter cuadragésimo que ya se movió. Este carácter sería el carácter número 41. Pero solamente hay 40 caracteres en una línea normal; que sucederá? Si hemos implementado movimiento grueso, el carácter 41 aparecerá repentinamente sobre la pantalla, después que el primer carácter haya desaparecido por el lado izquierdo. Esta aparición brusca es desagradable. La solución a este problema ya se encuentra incluída en los circuitos. Hay tres opciones de despliegue para anchos de línea: el campo angosto (128 compases de color de ancho) , el campo normal (160 compases de color de ancho) y el campo amplio (192 compases de color de ancho). Estas opciones se escogen poniendo los bits correspondientes en el registro DMACTL. Al usar movimiento horizontal fino, ANTIC automaticamente escoge más datos de RAM de los que despliega. Por ejemplo, si DMACTL se pone para campo normal, lo que en modo BASIC 0 corresponde a 40 bytes por línea, ANTIC de hecho tomará datos a una tasa apropiada para un campo ancho ---48 bytes por línea. Esto significará pérdida de datos horizontalmente si no se toma en cuenta. El problema no se manifiesta si el programador ya ha organizado la RAM de pantalla en líneas horizontales largas tal como se muestra en la Figura 6.1.

El problema correspondiente al movimiento vertical puede manejarse en dos formas. La forma descuidada es ignorarlo. En esa forma no tendremos líneas por mitades en dos extremos de la pantalla. Al contrario, las imágenes de la parte de abajo del despliegue no se moverán apropiadamente; ellas aparecerán repentinamente a la vista. La forma correcta toma muy poco trabajo. Para lograr un movimiento fino apropiado al entrar a

y al salir de la región de despliegue, debemos dedicar una línea de modo a actuar como memoria de reserva. Logramos esto absteniéndonos de poner el bit de movimiento vertical en la instrucción de lista de despliegue de la última línea de modo de la zona de movimiento vertical. La ventana se moverá ahora sin los saltos desagradables. La imagen de pantalla se habrá reducido en una línea de modo. Se hace notoria ahora la ventaja de los despliegues con movimiento. Es muy posible crear imágenes de pantalla que tengan más de 192 líneas de barrido en el despliegue. Esto podría resultar desastroso con un despliegue estático, pero con un despliegue en movimiento, las imágenes que están por sobre y por debajo de la región desplegada siempre pueden moverse para llevarlos a la vista.

Otra vez el movimiento fino sólo podrá llegar hasta aquí. El límite vertical para movimiento fino es de 16 líneas de barrido. Si tratamos de lograr movimiento más allá de este límite, ANTIC simplemente ignorará los bits superiores de los registros de movimiento. Para obtener movimiento fino vertical total (caso en el cual la pantalla se moverá suavemente tanto como nosotros queramos), debemos acoplar el movimiento fino con el grueso. Para hacer esto, primero hacemos un movimiento suave de la pantalla llevando la cuenta de cuánto la hemos movido. Una vez que la cantidad de movimiento fino iguale el tamaño del pixel, reponemos el registro de movimiento fino en 0 y ejecutamos,el movimiento grueso. La Figura 6.2 ilustra el proceso:

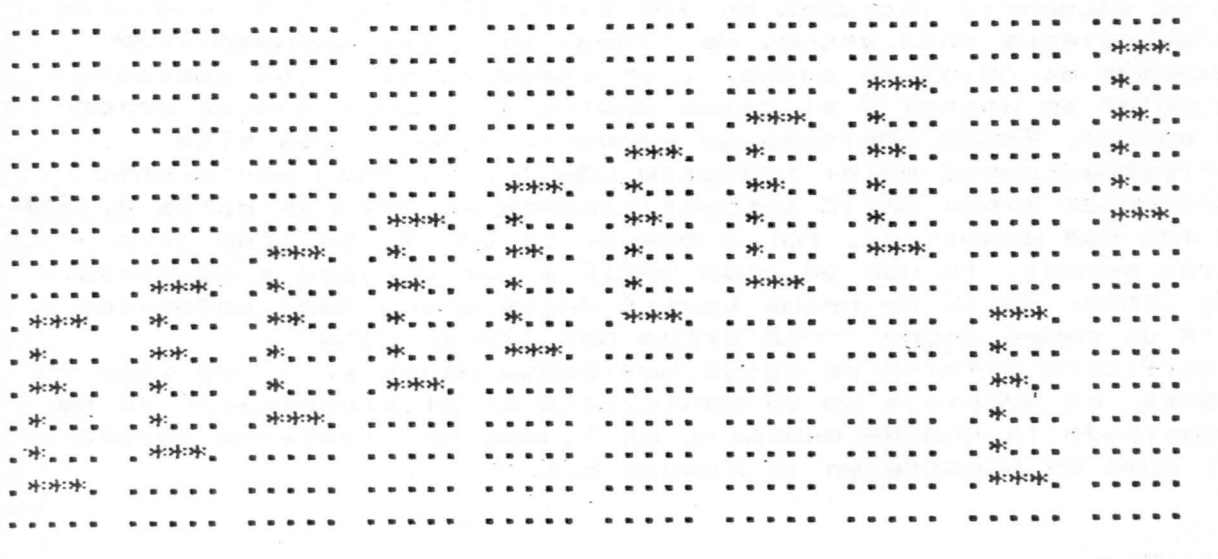


Figura 6.2
Combinando movimientos verticales fino y grueso.

El siguiente programa ilustra un movimiento fino simple.

```
1 HSCROL=54276
2 VSCROL=54277
10 GRAPHICS 0:LIST
20 LDES=PEEK(560)+256*PEEK(561)
30 POKE LDES+10,50:REM          habilitar ambos movimientos
                                para dos líneas
40 POKE LDES+11,50:REM
50 FOR Y=0 TO 7
60 POKE VSCROL,Y:REM            movimiento vertical
70 GOSUB 200:REM                retardo
80 NEXT Y
90 FOR X=0 TO 3
100 POKE HSCROL,X:REM           movimiento horizontal
110 GOSUB 200:REM               retardo
120 NEXT X
130 GOTO 40
200 FOR T=1 TO 200
210 NEXT T:RETURN
```

Este programa muestra un movimiento fino a muy baja velocidad. Hace evidentes varios problemas, que aparecen al usar el movimiento fino. Primero, las líneas de despliegue bajo la ventana de movimiento están desplazadas hacia la derecha. Esto se debe a que ANTIC automaticamente toma 48 bytes por línea en vez de los 40.
El problema solamente aparece en programas demostrativos irreales tal como éste; en aplicaciones verdaderas el agrupamiento de los datos de pantalla (tal como se muestra en la Figura 6.1) lo evitará. El segundo, y que es más serio, aparece cuando los registros de movimiento se modifican mientras ANTIC se encuentra en un proceso de despliegue. Ello confunde a ANTIC y hace que la pantalla salte. La solución consiste en modificar los registros de movimiento solamente durante los períodos de borrado vertical. Ello puede lograrse sólo con rutinas de lenguaje assembler. Por ello, el movimiento fino normalmente requiere del uso de este lenguaje.

APLICACIONES

Las aplicaciones gráficas de movimiento fino total son múltiples. Obvia es la utilizada en mapas grandes creados con gráficas de caracteres. Recurriendo al modo gráfico BASIC 2, Chris Crawford (autor de "Eastern Front", un clásico de los juegos de estrategia.) creó un enorme mapa de Rusia, que

contiene alrededor de 10 pantallas de imagen. La pantalla se convierte en una ventana de este mapa. El usuario puede moverse a través de todo el mapa con un bastón de control. El sistema es muy eficiente en cuanto a memoria; todo el programa de mapa, más los datos, más la lista de despliegue, más la redefinición del juego de caracteres, requiere un total de alrededor de 4Kbyte de RAM.

Hay muchas otras aplicaciones de esta técnica. Cualquier imagen muy grande trazada con gráfica de caracteres se presta para ser manejada por este sistema (El movimiento en sí no requiere de gráficas de caracteres. Las gráficas de mapa son menos deseables para aplicaciones de movimiento por sus grandes requerimientos de memoria). Podrían representarse por esta vía circuitos esquemáticos electrónicos de gran tamaño. Puede recurrirse al bastón de control para el movimiento a lo largo y ancho del diagrama y para indicar componentes particulares que quiera especificar el usuario. También planos o diagramas de arquitectura podrían desplegarse con esta técnica. Cualquier imagen grande, que no sea necesaria de ver en su integridad de una vez, podría presentarse con este sistema.

También puede usarse con bloques de texto de gran tamaño, a pesar de que pueda no resultar práctico leer bloques continuos de texto moviendo la imagen. El sistema se presta más para presentar bloques de texto independientes. Una idea particularmente excitante es aplicar el sistema a los menús. El programa parte presentando algún signo de bienvenida en la pantalla con símbolos que indiquen los submenús que apunten hacia otras regiones de una imagen mayor. 'Por aquí para sumar' podría apuntar hacia arriba mientras 'por acá para restar' podría apuntar hacia abajo. El usuario se mueve a través del menú con su bastón, persiguiendo las opciones. Cuando desee hacer una elección, ubica el cursor sobre la opción y presiona el botón de disparo. Aunque este sistema no necesariamente puede aplicarse a todo tipo de programa, podría ser de gran valor en algunos casos.

Hay dos aplicaciones un tanto disparadas del movimiento fino que aún no se han explorado bien. La primera es el movimiento fino selectivo, en el cual líneas de diferentes modos del despliegue podrían tener diferentes bits de movimiento habilitado.
Normalmente nos gustaría que toda la pantalla se moviera, pero no necesariamente será siempre así. Podríamos elegir una línea solamente para movimiento horizontal, otra para movimiento vertical y así sucesivamente. El segundo disparo apunta hacia la posibilidad de interrupciones de lista de despliegue, modificando al vuelo los registros HSCROL o VSCROL. La modificación de VSCROL durante el proceso de trazado es una operación muy intrincada, probablemente confundiría a ANTIC y

producirìa resultados no deseados. La modificación de HSCROL igualmente es intrincada pero podrìa ser más fácil.

CAPITULO 7

RESUMEN GENERAL de BASIC ATARI

Este capitulo discute el BASIC ATARI. Las cuatro materias principales son:

1. Qué es BASIC ATARI - una descripción de lo que se requiere para hacer funcionar BASIC, y una discusión de sus fuertes y debilidades.

2. Cómo opera BASIC ATARI - un análisis detallado de como se codifican los programas y como se ejecutan.

3. Mejorando el desempeño de los programas - una lista de metodos para aumentar la velocidad de un programa y disminuir su tamaño.

4. Técnicas de programación avanzada - una serie de aplicaciones especificas o trucos para satisfacer varias necesidades de programación.

Qué es BASIC ATARI

BASIC ATARI es como otros BASICs en el sentido de que es un lenguaje interpretado. Esto significa que los programas pueden correrse a medida que se ingresan, sin la necesidad de etapas intermedias de compilación o de enlace. El intérprete de BASIC ATARI reside en un cartridge de ROM de 8 Kbytes, en la ranura (izquierda) del computador. Engloba las direcciones A000 hasta BFFFF. El BASIC almacena los programas de usuario en RAM, para lo cual se requiere un mínimo de 8 Kbytes.

Existen actualmente tres versiones del BASIC ATARI. El primero de ellos, llamado versión A, es el que en forma de cartridge se emplea con los computadores modelo 400, 800 y 1200XL. La versión B es la que se encuentra incluìda en los modelos 600XL y 800XL. Ellos deberían tener las mismas características y el mismo comportamiento frente al usuario, además de ser compatibles entre sí. Sin embargo, la condición de compatibilidad sólo se da en forma completa, si se observan las siguientes reglas, muy simples y que de ninguna manera constituyen restricción, para evitar los problemas que podrían presentarse en caso contrario.

En esencia, el problema consiste en el bloqueo del computador provisto de una determinada versión del BASIC, cuando en él se quiere ejecutar un programa creado bajo la otra versión. El bloqueo solamente se presenta, y sólo en algunos casos, cuando el programa se ha grabado y recuperado en formato ATASCII (con LIST y ENTER respectivamente). Para evitarlo entonces basta, guardar y recuperar los programas en forma codificada (con SAVE y LOAD respectivamente, cuando se espera transportarlo entre computadores, que empleen versiones de BASIC diferentes.

Una segunda alternativa de solución, cuando sea ineludible el uso de LIST y ENTER, es la inhibición del BASIC original (versión B) del computador 600XL u 800XL, insertando un cartridge BASIC (versión A).

A raíz del problema descrito, ATARI ha preparado una nueva versión de BASIC, la C, que no presenta incompatibilidades ni con la versión A ni con la B.

Para usar el BASIC ATARI en buena forma, unc debe conocer sus fuerzas y sus debilidades. Dada esta información se pueden escribir programas que hagan un buen uso de las virtudes y ventajas del BASIC ATARI.

Los fuertes del BASIC ATARI son:

1. Soporta la gráfica del sistema operativo - se pueden hacer llamados simples de gráfica para desplegar información sobre la pantalla.

2. Soporta la circuitería - llamadas como SOUND, STICK y PADDLE constituyen simples interfaces a la circuitería del computador.

3. Una interfaz simple hacia el lenguaje assembler - la función USR da al usuario fácil acceso a rutinas de lenguaje assembler.

4. Intérprete en ROM - el intérprete BASIC está en ROM, lo cual previene modificaciones accidentales por parte del programa de usuario.

5. Soporte del DOS - llamadas especializadas como NOTE y POINT (DOS 2.0S y 3.0) permiten al usuario acceso aleatorio al disco, a través de su sistema operativo.

6. Soporte de periféricos - cualquier periférico reconocido por el sistema operativo, puede accederse también desde un programa en BASIC.

Debilidades del BASIC ATARI:

1. No soporta integros - todos los números se almacenan como números BCD de coma flotante de 6 bytes.

2. Paquete matemático lento - como todos los números son de seis bytes, las operaciones matemáticas son más bien lentas.

3. No hay agrupamientos de strings - solamente pueden crearse strings unidimensionales.

Cómo trabaja el BASIC ATARI

Un vistazo general breve a la forma de operar del intérprete BASIC es como sigue:

1. BASIC recibe una línea de ingreso del usuario y la convierte a su forma codificada.

2. A continuación pone esta línea en un programa de códigos.

3. A continuación este programa se ejecuta.

Los detalles de estas operaciones se discuten en las siguientes cuatro secciones.

A. EL PROCESO DE CODIFICACION

B. LA ESTRUCTURA DEL ARCHIVO DE CODIGOS

C. EL PROCESO DE EJECUCION DEL PROGRAMA

D. INTERACCION CON EL SISTEMA

A. EL PROCESO DE CODIFICACION

En términos simples, la codificación de una línea en BASIC es como sigue:

1. BASIC recibe una línea de ingreso.
2. Verifica la legalidad de su sintaxis.
3. Durante la verificación de sintaxis se codifica la línea.
4. La línea codificada se mueve hacia el programa de códigos.
5. Si la línea es de modo inmediato, se ejecuta.

Para entender en mejor forma el proceso de codificación, deben definirse inicialmente algunos términos. Ellos son:

Código (Token)	un byte de 8 bits que contiene un código particular suceptible de ser interpretado.
Sentencia	una expresión completa de códigos, que hacen que BASIC realice algunas tareas con sentido. En formato LIST las sentencias se separan por dobles puntos.
Línea	una o más sentencias precedidas ya sea por un número de línea comprendido en el rango de 0 a 32767, o una línea de modo inmediato sin número.
Comando	el primer código ejecutable de una sentencia que indica a BASIC que debe interpretar los códigos que sigan en una forma particular.
Variable	código que es un puntero indirecto a su valor real; así el valor puede alterarse sin cambiar el código.
Constante	un valor BCD de 6 bytes precedido por un código especial. Este valor permanece invariable a través de toda la ejecución del programa.
Operador	cualquiera de los 46 códigos que de una forma u otra modifican los valores que le siguen.
Función	un código, el cual al ser ejecutado retorna un valor al programa.
EOL	fin de línea (End of Line), un carácter con el valor hexadecimal 9B.

BASIC inicia el proceso de codificación, recibiendo una línea de ingreso. Este ingreso se obtendrá de uno de los administradores (handlers) del sistema operativo. Normalmente procederá del editor de pantalla, sin embargo, con el comando ENTER, puede especificarse cualquier dispositivo. El llamado emitido por BASIC es un comando GET RECORD, y el dato retornado es una información ATASCII terminada en un EOL. El sistema central de entrada/salida (CIO) almacena estos datos en la memoria de transferencia de ingreso de BASIC, ubicada entre 580 y 5FF hexadecimal.

Una vez recibido el registro, se inician los procesos de comprobación de sintaxis y de codificación. Primero BASIC busca un número de línea. Si lo encuentra, lo convierte en un número integro de dos bytes. Si no hay número de línea, se supone que se trata de un modo inmediato y se le asigna el número de línea 8000 hexadecimal. Estos serán los dos primeros códigos de la línea codificada. La línea se va construyendo en la memoria de transferencia de salida, que tiene 256 bytes de largo y se encuentra al final del área reservada de la RAM del sistema operativo.

El siguiente código es un byte de relleno, reservado para la cuenta de bytes (o el desplazamiento) a partir del comienzo de esta línea hasta el comienzo de la línea siguiente. A continuación hay otro byte de relleno para la cuenta desde el comienzo de la línea hasta el comienzo de la siguiente sentencia. Estos valores se pondrán en definitiva, una vez que se haya completado la codificación de la línea y de la sentencia respectivamente. El uso de estos valores se discute en la sección del proceso de ejecución del programa.

BASIC ahora busca el comando de la primera sentencia de la línea de ingreso. Se hace una comprobación para determinar si corresponde a un comando válido, recorriendo una lista de comandos legales que se encuentra en ROM. Si se encuentra una coincidencia, el byte que se ubica ahora en la línea de codificación, corresponde al número de ingreso en la lista de ROM que coincida. Si no hay coincidencia, se asigna un código de error de sintaxis a este byte y BASIC detiene la codificación, copia el resto de la línea de ingreso en formato ATASCII a la memoria de transferencia de salida de códigos e imprime la línea de error.

Suponiendo una línea correcta, el comando siguiente puede ser uno de siete: una variable, una constante, un operador, una función, una comilla, otra sentencia o un EOL. Basic verifica ahora si el siguiente carácter de ingreso es de tipo numérico. De no ser así, lo compara y los que siguen contra los ingresos de la tabla de nombres de variables. Siendo ésta la primera línea de código del programa que se ha ingresado, no habrá coincidencia. Se comparan a continuación los caracteres contra las tablas de funciones y de operadores. Si nuevamente

no hay coincidencia, BASIC supone que se trata del nombre de una variable, nueva. Como es la primera variable se le asignará la primera posición en la tabla de nombres de variables. Los caracteres se copian desde la memoria de transferencia de ingreso y se almacenan en la tabla de nombres con el bit más significativo (MSB) puesto en el último byte del nombre. Enseguida se reservan 8 bytes de la tabla de valores de variables para este ingreso (Vea la discusión sobre la tabla de valores de variables en la sección de estructura de archivos codificados).

El código que finalmente aparece en la línea codificada es el número de orden de la variable menos uno, con el bit más significativo (MSB) puesto. Así el código de la primera variable ingresada sería 80 hexadecimal, el segundo 81 y así hasta FF con un total de 128 números definidos para variables.

Si se trata de una función, se le asigna como código el número de orden de esa función en la tabla de funciones de operador. Las funciones requieren determinadas secuencias de párametros; ellas se encuentran en las tablas de sintaxis y, si no se produce la coincidencia necesaria, resultará un error de sintaxis.

Si se trata de un operador, se le asigna un código a su número de orden en la tabla de ingresos. Los operadores pueden sucederse en forma bastante compleja (como en el caso de paréntesis múltiples) de modo que la verificación de sintaxis es un tanto complicada.

En el caso de comillas, BASIC supone que sigue una cadena de caracteres (string), asigna un 0F hexadecimal al código de salida y reserva un byte de relleno para el largo del string. Los caracteres se transfieren de la memoria de entrada hacia la memoria de salida hasta que se encuentra el segundo par de comillas. A continuación se pone el byte correspondiente al largo de la cadena de caracteres.

Si los caracteres que siguen en la memoria de entrada son numéricos, BASIC los convertirá en una constante BCD (decimal codificado en binario), de seis bytes. En la memoria de salida se pondrá un código hexadecimal 0E, seguido de la constante de seis bytes.

Cuando se encuentra un doble punto, se inserta un código 14 hexadecimal en la memoria de salida y el desplazamiento con respecto al comienzo de la línea se almacena en el byte de relleno reservado para la cuenta hasta el comienzo de la sentencia siguiente. En este punto se hace una reserva de otro byte de relleno y el proceso vuelve para obtener un nuevo comando.

Al encontrar un EOL se almacena un código 1E hexadecimal

y el deaplazamiento desde el comienzo de la línea se almacena en el byte de relleno, reservado para ello. A esta altura, el proceso de codificación se ha completado y BASIC moverá la línea codificada hacia el programa codificado. En primer lugar revisa el programa para ver si este número de línea ya existe, si es así, la línea nueva reemplaza a la antigua. Si no existe una línea con ese número, la nueva línea se inserta en la correcta secuencia numérica. En ambos casos, los datos que sigan a la línea se moverán, ya sea hacia adelante o hacia atrás en la memoria, para dar cuenta de la expansión o contracción del programa.

BASIC ahora verifica si la línea codificada corresponde a una línea de modo inmediato. Si es así, la línea se ejecuta de acuerdo a los métodos descritos en el proceso de interpretación; si no, BASIC retorna para recibir otra línea de ingreso.

Si en cualquier instante durante el proceso de codificación, el largo de la línea codificada excede de 255 bytes, se envía un mensaje de error 14 (línea demasiado larga) a la pantalla y BASIC retorna para recibir la siguiente línea de ingreso.

Un ejemplo de línea de ingreso y su forma codificada, tiene la siguiente apariencia (todos los valores de código son hexadecimales):

```
                    10 LET X=1 : PRINT X

  0A 00 13 0F 0E 80 2D 0E 40 01 00 00 00 00 14 13 20 80 22
  ↑--+-↑ ↑  ↑  ↑  ↑  ↑  ↑  ↑  ↑-------+-------↑ ↑  ↑  ↑  ↑  ↑
    |   | |  |  |  |  |  |  |          |       | |  |  |  |  |
Línea 10| |  | Let |  =  |  |          1       | |  |  X Fin
      | | |  |  |  |     |                     | |  |    de
   despl.| |  X  |     |                     | |Print línea
     de  | |     Constante                     |despl.   (EOL)
   línea | |      numérica                     | de
         | |                                   |sentencia
         | |                                   |
      desplaz.                              fin de
      sentencia                             sentencia
```

COMANDOS

HEX	DEC	
00	0	REM
01	1	DATA
02	2	INPUT
03	3	COLOR
04	4	LIST
05	5	ENTER
06	6	LET
07	7	IF
08	8	FOR
0A	10	GOTO
0B	11	GO TO
0C	12	GOSUB
0D	13	TRAP
0E	14	BYE
0F	15	CONT
10	16	COM
11	17	CLOS
12	18	CLR
13	19	DEG
14	20	DIM
15	21	END
16	22	NEW
17	23	OPEN
18	24	LOAD
19	25	SAVE
1A	26	STATUS
1B	27	NOTE
1C	28	POINT
1D	29	XIO
1E	30	ON
1F	31	POKE
20	32	PRINT
21	33	RAD
22	34	READ
23	35	RESTORE
24	36	RETURN
25	37	RU
26	38	STOP
27	39	POP
28	40	?
29	41	GET
2A	42	PUT
2B	43	GRAPHICS
2C	44	PLOT
2D	45	POSITION
2E	46	DOS
2F	47	DRAWTO
30	48	SETCOLOR
31	49	LOCATE
32	50	SOUND
33	51	LPRINT
34	52	CSAVE
35	53	CLOAD
36	54	[LET IMPLICITO]
37	55	ERROR- [SINTAXIS]

OPERADORES

HEX	DEC	
0E	14	[CONST.NUM.]
0F	15	[CONST.STR.]
10	16	"
11	17	[SIN USO]
12	18	,
13	19	$
14	20	: [FIN SENT.]
15	21	;
17	23	GOTO
18	24	GOSUB
19	25	TO
1A	26	STEP
1B	27	THEN
1C	28	#
1D	29	<= [NUMERICO]
1E	30	<>
1F	31	>=
20	32	<
21	33	>
22	34	=
23	35	^
24	36	*
25	37	+
26	38	-
27	39	/
28	40	NOT
29	41	OR
2A	42	AND
2B	43	(
2C	44	)
2D	45	= [ASIGN.ARITM.]
2E	46	= [ASIGN.STRING]
2F	47	<= [STRINGS]
30	48	<>
31	49	>=
32	50	<
33	51	>
34	52	=
35	53	+ [UNARIO]
36	54	-
37	55	([PAR.IZQ.STRING]
38	56	([PAR.IZQ.AGRUP.]
39	57	([PAR.IZQ.DIM.AGRUP.]
3A	58	([PAR.IZQ.FUNCION]
3B	59	([PAR.IZQ.DIM.STRING]
3C	60	, [COMA AGRUP.]

FUNCIONES

HEX	DEC	
3D	61	STR$
3E	62	CHR$
3F	63	USR
40	64	ASC
41	65	VAL
42	66	LEN
43	67	ADR
44	68	ATN
46	70	PEEK
47	71	SIN
48	72	RND
49	73	FRE
4A	74	EXP
4B	75	LOG
4C	76	CLOG
4D	77	SQR
4E	78	SGN
4F	79	ABS
50	80	INT
51	81	PADDLE
52	82	STICK
53	83	PTRIG
53	84	STRIG

B. LA ESTRUCTURA DEL ARCHIVO DE CODIGOS

El archivo de códigos contiene dos grandes partes: 1) un grupo de punteros de página 0 que apuntan hacia el archivo de códigos, y 2) el archivo de códigos propiamente tales. Los punteros de página 0 son valores de dos bytes que apuntan a varias secciones del archivo de códigos. Hay nueve punteros de dos bytes que se encuentran en las ubicaciones 80 a 91 hexadecimal. La siguiente es una lista de estos punteros y de las secciones del archivo de códigos a las que apuntan.

PUNTERO (hex.)	SECCION DEL ARCHIVO DE CODIGOS (Bloques contiguos)
LOMEM 80,81	Memoria de salida de códigos - es ésta la memoria de transferencia que usa BASIC para codificar cada línea de código. Tiene un largo de 256 bytes. La memoria se encuentra al final de la RAM asignada al sistema operativo.
VNTP 82,83	Tabla de nombres de variables - una lista de todos los nombres de variables que han sido ingresadas en el programa. Se almacenan en caracteres ATASCII, cada nombre almacenado en el mismo orden en que fue ingresado. Existen tres tipos de nombres: 1. Variables escalares - con el bit más significativo (MSB) puesto en el último carácter del nombre. 2. Variables string - el último carácter es un "$" con el MSB puesto. 3. Variables de agrupamiento - el último carácter es un "(" con el MSB puesto.
VNTD 84,85	Fin de relleno de la tabla de nombres de variables - BASIC usa este puntero para indicar el fin de la tabla de nombres. Este normalmente apunta a un byte de relleno de valor 0 cuando hay menos de 128 variables. Cuando hay 128 variables apunta al último byte del último nombre de variable.
VVTP 86,87	Tabla de valores de variables - esta tabla contiene la información actual de cada variable. Para cada variable en la tabla de nombres, se reservan 8 bytes en la tabla de valores. La información para cada tipo de variables es la siguiente:

BYTE No.	1	2	3	4	5	6	7	8
ESCALAR	00	#Var	Constante BCD de 6 Bytes					
AGRUP. (DIM)	41	#Var	Despl. c/r a STARP (8C,8D)		primera DIM + 1		segunda DIM + 1	
(no DIM)	40							
STRING (DIM)	81	#Var	Despl. c/r a STARP (8C,8D)		LARGO		DIM	
(no DIM)	80							

Una variable escalar contiene un valor numérico. Un ejemplo es X=1. El escalar es X y su valor es 1, almacenado en formato BCD de seis bytes. Un agrupamiento se compone de elementos numéricos almacenados en el área de strings/agrupamientos y tiene una posición en la tabla de valores. Un string compuesto de caracteres en el área de strings/agrupamientos también tiene una posición en la tabla.

El primer byte de cada ingreso de valor indica el tipo de variable: 00 para un escalar, 40 para un agrupamiento y 80 para un string. Si el agrupamiento o string ya ha sido dimensionado, se pone el bit menos significativo (LSB) del primer byte.

El segundo byte contiene el número de la variable. La primera variable ingresada corresponde al número 0 y si hay 128 variables, la última tiene el número $7F.

En el caso de la variable escalar, los bytes tercero a octavo contienen el numero BCD de seis bytes, que en la actualidad le ha sido asignado.

Para los agrupamientos y strings, los bytes tercero y cuarto contienen el desplazamiento del comienzo de los datos con respecto al comienzo del área de string/agrupamiento descrito más abajo.

Los bytes quinto y sexto de un agrupamiento contienen su primera

dimensión. La cantidad es un ìntegro de 2 bytes y su valor es mayor en 1 que el ingresado por el usuario. Los bytes séptimo y octavo corresponden a la segunda dimensión, también aumentado en una unidad.

Los bytes quinto y sexto de un string corresponden al ìntegro de dos bytes, que contiene su largo actual. Los bytes séptimo y octavo corresponden a su dimensionamiento.

STMTAB 88,89 — Tabla de sentencias - este bloque de datos incluye todas las líneas de códigos que han sido ingresadas por el usuario y codificadas por BASIC, también incluye la línea de modo inmediato. El formato de estas líneas se describe en el ejemplo de línea codificada dentro de la sección del proceso de codificación.

STMCUR 8A,8B — Sentencia actual - este puntero es usado por BASIC para indicar códigos particulares dentro de una línea de la tabla de sentencias. Cuando BASIC se encuentra esperando un ingreso, este puntero se apunta al comienzo de la línea de modo inmediato.

STARP 8C,8D — Area de string/agrupamientos - este bloque contiene todos los datos de strings y de agrupamientos. Los caracteres de un string se almacenan como ingresos ATASCII de un byte, de modo que un string de 20 caracteres requerirá 20. bytes. Los agrupamientos se almacenan a través de números BCD de seis bytes para cada elemento. Así un agrupamiento de 10 elementos requerirá 60 bytes.

Este área se asigna y subsecuentemente se incrementa con cada sentencia de dimensionamiento encontrada, siendo la cantidad en cada caso igual a la magnitud de la dimensión del string o seis veces la magnitud de la dimensión de un agrupamiento.

RUNSTK 8E,8F — Stack de ejecución - este stack de software contiene ingresos GOSUB y FOR/NEXT. Un ingreso GOSUB consiste de cuatro bytes. El primero es un byte de valor 0 que significa GOSUB, seguido de un número de línea de dos bytes que

corresponde al lugar donde se produjo el llamado, seguido del desplazamiento dentro de la línea, de modo que el RETURN pueda volver y ejecutar la sentencia siguiente.

El ingreso FOR/NEXT es de 16 bytes. El primer valor es el límite que puede alcanzar la variable de contador. El segundo es el paso de incremento del contador. Cada una de estas cantidades se encuentra en formatos BCD de 6 bytes. El byte décimotercero es el número de la variable del contador con su bit más significativo (MSB) puesto; los bytes décimocuarto y décimoquinto corresponden al número de línea, mientras el décimosexto da el desplazamiento de la sentencia FOR dentro de la línea.

MEMTOP 90,91

Extremo alto de la RAM de aplicación - corresponde al final del programa de usuario. La expansión del programa se realizará a partir de este punto y hasta el final de la memoria RAM libre, la que se define por el comienzo de la lista de despliegue. La función FRE retorna la cantidad de RAM libre, restando MEMTOP de HIMEM (2E5,2E6). Note que este MEMTOP BASIC no es el mismo que la variable del sistema operativo llamada MEMTOP.

C. EL PROCESO DE EJECUCION DEL PROGRAMA

El proceso de ejecución de una línea de código comprende la lectura de los códigos que se crearon durante el proceso de codificación. Cada código tiene un significado particular, que hace que BASIC ejecute una serie de operaciones específicas. El método de hacerlo requiere que BASIC reciba un código por vez, del programa de códigos y que lo procese a continuación. El código constituye un índice a una tabla de saltos a rutinas, de modo que un código PRINT apuntará indirectamente a una rutina de procesamiento PRINT. Una vez completo este procesamiento, BASIC retorna para recibir el código siguiente. El puntero usado para tomar cada código se llama STMCUR y se encuentra en 8A y 8B.

La primera línea de código de un programa que se ejecuta es la línea de modo inmediato. Se trata generalmente de un RUN o de un GOTO. En el caso del RUN, BASIC recibe la primera línea de códigos de la tabla de sentencias (programa codificado) y la procesa. Si todo el código es secuencial, BASIC simplemente ejecutará líneas consecutivas.

Al encontrar un GOTO debe encontrarse primero la línea referida. La tabla de sentencias contiene una lista parcialmente enlazada de números de líneas y sentencias, comenzando por el número más bajo, seguida de números de líneas crecientes hasta la mayor. Si se necesita una línea de algún lugar intermedio de la tabla, el proceso es el siguiente:

> La dirección de la primera línea se encuentra en el puntero STMTAB de 88 y 89. El valor se almacena en un puntero temporal. Los primeros dos bytes de la primera línea son su número de línea. Este número se compara con el número de línea buscado. Si el primer número es menor, BASIC va a la línea siguiente, agregando el tercer byte de la primera línea al puntero temporal. El puntero temporal estará apuntando ahora a la segunda línea. Nuevamente los dos primeros bytes de esta nueva línea se comparan con la línea buscada y si son menores, se agrega el tercer byte al puntero. Si se produce una coincidencia de número de línea, el contenido del puntero temporal se mueve a STMCUR y BASIC buscará el código siguiente de esta nueva línea. En caso de no encontrarse el número de línea buscado, se genera un error 12.

GOSUB requiere de un procesamiento mayor que GOTO. La rutina que encuentra la línea es la misma, pero BASIC, antes de ir a esta línea, establece un ingreso en el stack de ejecución. Asigna cuatro bytes al final del stack y almacena un 0 en el primer byte, para indicar que se trata de un ingreso de stack GOSUB. Enseguida almacena el número de línea en el cual se encontraba cuando se hizo el llamado, en los dos bytes siguientes del stack. El último byte contiene el

desplazamiento, expresado en bytes, del código GOSUB, con respecto al comienzo de la línea en la cual se encuentra. BASIC a continuación ejecuta la línea buscada. Al encontrar un RETURN, se bota el ingreso del stack y BASIC retorna a la línea del llamado.

El comando FOR hace que BASIC asigne 16 bytes en el stack de ejecución. Los primeros seis bytes corresponden al límite que la variable puede alcanzar en formato BCD de seis bytes. Los siguientes seis bytes corresponden al paso expresado en el mismo formato. A continuación BASIC almacena el número de variable (con sus MSB puesto) de la variable contador. A continuación almacena el número de línea actual (2 bytes) y el desplazamiento dentro de la línea. A continuación se ejecuta el resto de la línea.

Cuando BASIC encuentra el comando NEXT revisa el último ingreso del stack. Se asegura que la variable referida FOR/NEXT es la misma que se encuentra en el stack y comprueba si el contador ha alcanzado o excedido sus límites. En caso negativo, BASIC retorna a la línea de la sentencia FOR y continúa la ejecución. Si ya se ha alcanzado el límite se bota el ingreso FOR del stack y la ejecución continúa a partir de ese punto.

Al evaluar una expresión, los operadores se ponen en el stack de operadores y se sacan de él de a uno para ser evaluados. El orden en el cual los operadores se ponen en el stack puede ser ya sea implícito, caso en el cual BASIC averigua la precedencia de operadores a partir de una tabla en ROM, o el orden puede darse explícitamente por la ubicación de paréntesis.

Si en algún momento se presiona la tecla BREAK, el sistema operativo pone una bandera (flag) para indicar este hecho. BASIC revisa esta bandera después del procesamiento de cada código. Si la encuentra puesta, almacena el número de línea en que ocurrió, imprime un mensaje "STOPPED AT LINE número", borra la bandera BREAK y espera un ingreso de usiario. En este punto puede digitarse CONT, haciendo que la ejecución del programa se reanude en la línea siguiente.

D. INTERACCION CON EL SISTEMA

BASIC se comunica con el sistema operativo principalmente a través del uso de las llamadas de I/O (entrada/salida) al servicio central de entrada/salida (CIO). A continuación se presenta una lista de los llamados BASIC de usuario y los correspondientes bloques de control entrada/salida (IOCB) del sistema operativo.

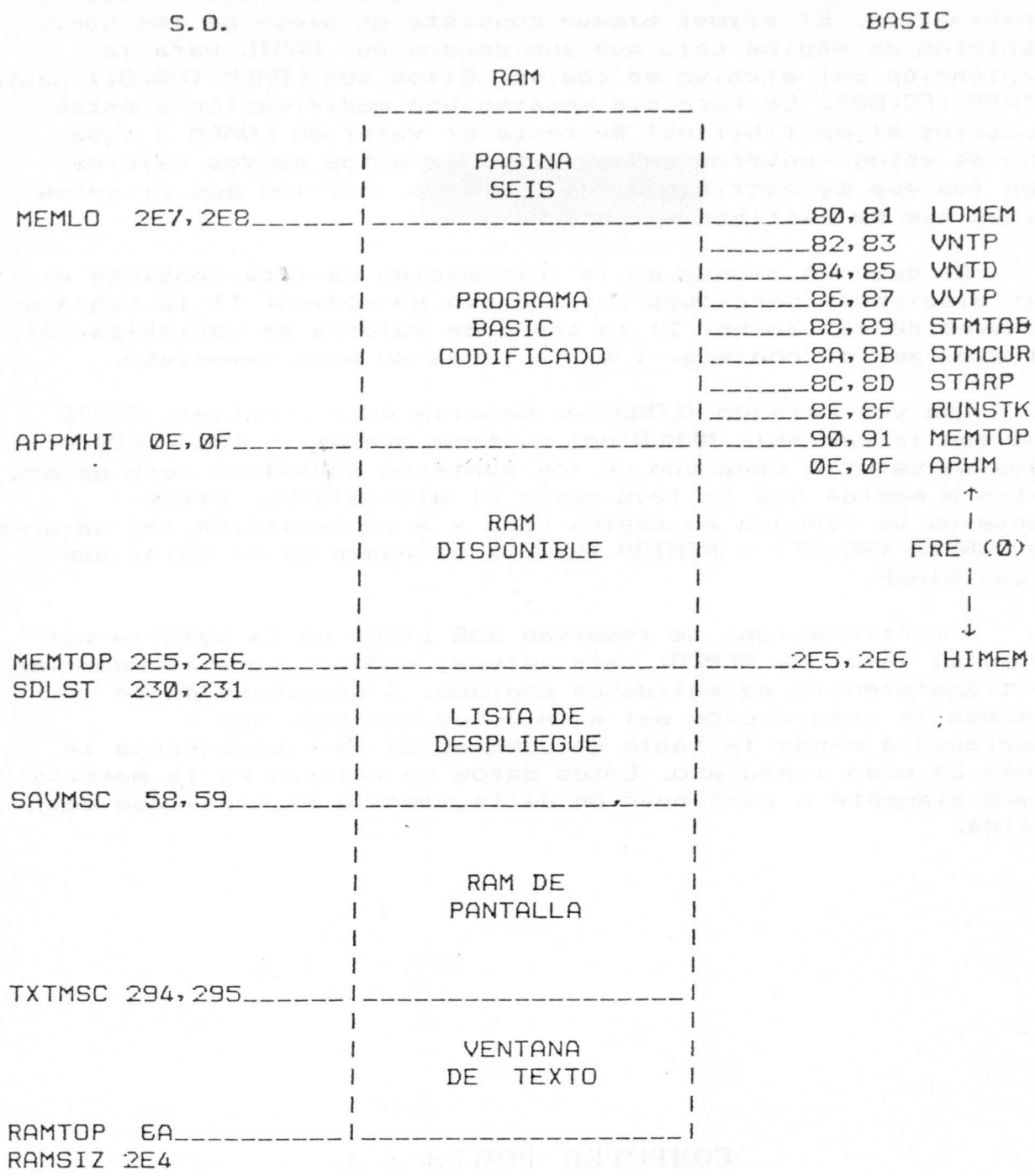
BASIC	SO
OPEN #1,12,0,"E:"	IOCB=1 COMANDO=3 (OPEN) AUX1=12 (Entrada/Salida) AUX2=0 Dirección memoria transpaso=ADR("E:")
GET #1,X	IOCB=1 Comando=7 (Get Characters, recibir caracteres) Largo de memoria de transpaso=0 El carácter se retorna al acumulador
PUT #1,X	IOCB=1 Comando=11 (Put Characters, poner caracteres) Largo de memoria transpaso=0 Salida del carácter a traves del acumulador
INPUT #1,A$	IOCB=1 Comando=5 (Get Record, recibir registro) Largo de la memoria de transpaso=largo de A$ (no más de 120) Dirección memoria transpaso=memoria de línea de ingreso
PRINT #1,A$	IOCB=1 BASIC emplea un vector especial 'poner byte' del IOCB para hablar directamente al administrador (handler).
XIO 18,#6,12,0,"S:"	IOCB=6 Comando=18 (especial-rellenar) Aux1=12 Aux2=0

SAVE/LOAD: Cuando se salva o graba (SAVE) un programa de BASIC codificado a un dispositivo, se escriben dos bloques de información. El primer bloque consiste de siete de los nueve punteros de página cero que son usados por BASIC para la mantención del archivo de código. Ellos son LOMEM (80,81) hasta STARP (8C,8D). Se hace sin embargo una modificación a estos punteros al escribirlos: Se resta el valor de LOMEM a cada uno de estos punteros de dos bytes, y estos nuevos valores son los que se escriben al dispositivo. Así los dos primeros bytes que se escriben serán 0,0.

El segundo bloque de la información escrita consiste de las siguientes secciones del archivo de código: 1) la tabla de nombres de variables, 2) la tabla de valores de variables, 3) el programa de códigos, y 4) la línea de modo inmediato.

Una vez cargado (LOAD) en memoria este programa, BASIC obtiene la variable MEMLO del sistema operativo (2E7,2E8) y suma su valor a cada uno de los punteros de página cero de dos bytes a medida que se leen desde el dispositivo. Estos punteros se reponen en página cero y a continuación los valores de RUNSTK (8E,8F) y MEMTOP (90,91) se ponen en el valor que tiene STARP.

A continuación, se reservan 256 bytes en la memoria por sobre el valor de MEMLO, para asignar espacio para la memoria de transferencia de salida de códigos. A continuación se ingresa la información del archivo de códigos, que comprenderá desde la tabla de nombres de variables hasta la línea de modo inmediato. Estos datos se colocan en la memoria, inmediatamente a continuación de la memoria de transpaso de salida.



PUNTEROS DEL S.O. Y DEL BASIC (sin DOS)

Mejoras en el desempeño de un programa

El desempeño de un programa puede mejorarse por dos vìas. Primero puede disminuirse el tiempo de ejecución (es decir, el programa correrá más rapidamente) y segundo, puede disminuirse la cantidad de memoria requerida, permitiendo un menor uso de RAM. Para conseguir estas dos metas, pueden usarse las siguientes listas como guìa de referencia. Los métodos de mejora en cada lista se han ordenado principalmente en orden de importancia decreciente. Por ello el método descrito al comienzo de la lista tendrá más impacto que él de más abajo.

Para acelerar la ejecución de un programa BASIC:

1. Reescriba el programa - como BASIC no es un lenguaje estructurado, sus instrucciones tienden a volverse ineficientes. Después de algunas revisiones, ello se vuelve aún peor. Asì, el tiempo gastado en reescribir las instrucciones bien vale la pena.

2. Verifique la lógica de los algoritmos - asegúrese que las instrucciones para ejecutar un determinado proceso sean tan eficientes como sea posible.

3. Ubique al comienzo del programa las subrutinas y los bucles FOR/NEXT que se llamen con mayor frecuencia - cuando BASIC busca un número de lìnea, inicia la búsqueda en el comienzo del programa, de modo que cualquier referencia a un punto próximo al final del programa requerirá una búsqueda más larga.

4. Operaciones llamadas frecuentemente desde el interior de un bucle deberìan estar en forma de instrucciones directas y no de subrutinas - la velocidad del programa puede crecer aquì ya que BASIC gasta algún tiempo agregando y sacando valores del stack de ejecución.

5. Haga que el bucle de mayores modificaciones dentro de un sistema anidado sea el más interno - asì el stack de ejecución se alterará en menor número de veces.

6. Simplifique los cálculos de coma flotante dentro de un bucle - si se obtiene un resultado multiplicando una constante por un contador, se puede salvar algo de tiempo cambiando la operación a una suma de una constante.

7. Construya los bucles como sentencias múltiples en una lìnea - asì el intérprete BASIC no tendrá que avanzar hasta la lìnea siguiente para continuar con el bucle.

8. Inhiba el despliegue de pantalla - si la información visual no es importante durante algun tiempo, puede disminuirse hasta un 30% el tiempo de ejecución, si se hace POKE 559,0.

9. Haga uso de un modo gráfico rápido o de una lista de despliegue corta - si no se requiere un despliegue de pantalla completo puede lograrse una reducción de hasta un 25% de tiempo.

10. Use código assembler - pueden lograrse ahorros de tiempo, codificando bucles en assembler y recurriendo a la función USR.

Ahorrando memoria en un programa BASIC:

1) Recodifique - como ya se dijo antes, reestructurando el programa lo hará más eficiente. También ahorrará memoria.

2. Elimine los REM - se almacenan como datos ATASCII y solamente consumen memoria de programa.

3. Reemplace las constantes, que se usen tres o más veces, por una variable. BASIC asigna siete bytes para una constante y solamente uno para una referencia a variable, así pueden ahorrarse seis bytes cada vez que una constante se reemplaza con una variable ya asignada a ese valor de constante.

4. Inicialice las variables con una sentencia READ - una sentencia data se almacena en código ATASCII, un byte por carácter, mientras que una sentencia de asignación requiere siete bytes para una constante.

5. Trate de convertir números que se usan una o dos veces a operaciones de variables predefinidas - un ejemplo sería definir Z1=1, Z2=2 y si se requiere el número 3, reemplazarlo por la expresión Z1+Z2.

6. Asigne números de línea que se usan con frecuencia (en GOSUB y GOTO) a variables predefinidas - si se hacen 50 referencias a la línea 100, pueden ahorrarse unos 300 bytes haciendo Z100 = 100 y usando referencias a Z100.

7. Mantenga en un mínimo el número de variables - cada nueva variable requiere 8 bytes adicionales en la tabla de valores de variables, más algunos bytes para su nombre.

8. Limpie las tablas de valores y de nombres - los ingresos de variables no se eliminan de las tablas de valores y de nombres por el solo hecho de haber eliminado referencias a ella del programa. Para eliminar los ingresos, liste el programa a disco o cassette, digite

NEW y enseguida entre el programa.

9. Mantenga los nombres de variables lo más cortos posibles - cada nombre de variable se almacena en la tabla de nombres como información ATASCII. Mientras más cortos sean los nombres, más corta será la tabla.

10. Reemplace todo texto usado con frecuencia con strings - en pantallas con mucho texto, puede ahorrarse memoria asignando un string a un conjunto de caracteres usado con frecuencia.

11. Inicialice los strings con sentencias de asignación - la asignación de un string con data entre comillas requiere menos espacio que una sentencia READ y una función CHR$.

12. Concatene líneas en sentencias múltiples - pueden ahorrarse tres bytes cada vez que dos líneas se conviertan en dos sentencias de una misma línea.

13. Reemplace las subrutinas que se empleen una sola vez por líneas directas - las sentencias GOSUB y RETURN constituyen un desperdicio de bytes si se usan una sola vez.

14. Reemplace agrupamientos numéricos con strings, si los valores de los datos no exceden de 255 - los ingresos de agrupamientos numéricos requieren 6 bytes cada uno, mientras un elemento de string sólo requiere 1.

15. Reemplace las sentencias SETCOLOR con comandos POKE, con ello se ahorrarán 8 bytes.

16. Recurra a los caracteres de control de cursor más que a la sentencia POSITION - la sentencia POSITION requiere 15 bytes para los parámetros X e Y, mientras los caracteres de edición de cursor requieren un byte cada uno.

17. Elimine líneas de código a través de control de programas - vea la sección de técnicas de programación avanzada.

18. Modifique el puntero de string/agrupamiento para cargar datos predefinidos - vea la sección de técnicas de programación avanzada.

19. Pueden almacenarse pequeñas rutinas assembler como sentencias REM - los REM se almacenan como datos inalterados en código ATASCII.

20. Concatene los programas - un ejemplo lo constituye una rutina de inicialización que se ejecuta primero, y la cual a su vez carga y ejecuta después el programa principal.

Técnicas de Programación Avanzada

Una vez entendidos los fundamentos de BASIC ATARI pueden escribirse algunas aplicaciones muy interesantes. Estas pueden ser operaciones estrictamente BASIC o pueden comprender también ingredientes del sistema operativo.

Ejemplo 1 - Inicialización de strings - Este programa dispondrá todos los bytes de un string de cualquier largo al mismo valor. BASIC copia el primer byte del string fuente al primer byte del string destino, enseguida el segundo, el tercero y así sucesivamente. Haciendo el string de destino igual al segundo byte de la fuente, puede almacenarse el mismo carácter en todo el string.

Ejemplo 2 - Eliminación de líneas de código - usando una característica del sistema operativo, un programa puede eliminar o modificar líneas de código dentro de si mismo. El editor de pantalla puede disponerse para aceptar datos de la pantalla sin ingresos por parte del usuario. Así, preparando primero una pantalla, ubicando a continuación el cursor en su rincón superior izquierdo y deteniendo el programa, BASIC recibirá los comandos que hayan sido impresos en la pantalla.

Ejemplo 3 - Grabar el área de strings/agrupamientos - Si un agrupamiento o string siempre se inicializa con los mismos datos y a la misma magnitud, puede ahorrarse un área apreciable de programa, grabando la información durante el SAVE y eliminando el código de inicialización de la ejecución siguiente.

Ejemplo 4 - Grabando los números en formato BCD en disco - cada vez que se escribe datos numéricos en un dispositivo, el envío se hace a través de información ATASCII. Esto significa que el nuúero 10 se escribe en ATASCII como 1 seguido de 0. Esto complica de sobremanera los registros de largo constante. Una forma de corregir esto, es almacenar los números BCD de seis bytes en el disco directamente, igualándolos a un string y escribiendo a continuación este string. Puede recuperárseles en la misma forma.

Ejemplo 5 - Gráficas de jugador/proyectil con strings - En este ejemplo se muestra una forma de mover con rapidez los datos de gráfica jugador/proyectil. A un string ya dimensionado se le modifica su valor de desplazamiento de área string/agrupamiento para que apunte al área de gráfica de jugador/proyectil. Escribiendo en este string con sentencias de asignamiento, permite escribir datos al área jugador/proyectil a una velocidad de lenguaje assembler.

```
10 REM Inicialización de un string
20 DIM A$(1000)
30 A$(1)="A":A$(1000)="A"
40 A$(2)=A$
```

```
10 REM Ejemplo de eliminación de línea
20 GRAPHICS 0:POSITION 2,4
30 ? 70:? 80:? 90:? "CONT"
40 POSITION 2,0
50 POKE 842,13:STOP
60 POKE 842,12
70 REM Estas líneas
80 REM van a ser
90 REM eliminadas
```

```
10 REM Grabación de string/agrupamiento
15 REM GOTO 20 en la primera ejecución
17 REM Eliminar línea 20 para la segunda pasada
18 GOTO 100
20 DIM A$(10):A$="WWWWWWWWWW"
30 STARP=PEEK(140)+256*PEEK(141)
40 STARP=STARP+10
50 HI=INT(STARP/256):LO=STARP-256*HI
60 POKE 140,LO:POKE 141,HI
70 SAVE "D:STRING":STOP
100 STARP=PEEK(140)+256*PEEK(141)
110 STARP=STARP-10
120 HI=INT(STARP/256):LO=STARP-256*HI
130 POKE 140,LO:POKE 142,LO:POKE 144,LO
140 POKE 141,HI:POKE 143,HI:POKE 145,HI
150 DIM A$(10)
160 A$(10,10)="W"
170 STOP
```

```
10 REM Grabar y recuperar números BCD con  disco
15 DIM A(0),B$(6)
20 B$(6,6)=CHR$(32)
30 VTAB=PEEK(134)+256*PEEK(135)
40 POKE VTAB+10,0
50 OPEN #1,8,0,"D:PRUEBA"
60 FOR C=1 TO 15:A(0)=C:? #1;B$:NEXT C
70 CLOSE #1
80 OPEN #1,4,0,"D:PRUEBA"
90 FOR C=1 TO 15:INPUT #1,B$:? A(0):NEXT C
100 CLOSE #1:END
```

```
100 REM Ejemplo de JUGADOR/PROYECTIL
110 DIM A$(512),B$(20)
120 X=X+1:READ A:IF A()-1 THEN B$(X.X)=CHR$(A):GOTO 120
130 DATA 0,255,129,129,129,129,129,129,129,129,255,0,-1
2000 POKE 559,62:POKE 704,88
2020 I=PEEK(106)-16:POKE 54279,I
2030 POKE 53277,3:POKE 710,224
2040 VTAB=PEEK(134)+256*PEEK(135)
2050 ATAB=PEEK(140)+256*PEEK(141)
2060 DESP=256*I+1024-ATAB
2070 HI=INT(DESP/256):LO=DESP-256*HI
2090 POKE VTAB+2,LO:POKE VTAB+3,HI
3000 Y=60:Z=100:V=1:H=1
4000 A$(Y,Y+11)=B$:POKE 53248,Z
4010 Y=Y+V:Z=Z+H
4020 IF Y)213 OR Y(33 THEN V=-V
4030 IF Z)206 OR Z(49 THEN H=-H
4420 GOTO 4000
```

CAPITULO 8
EL SISTEMA OPERATIVO

8.1 INTRODUCCION AL SISTEMA OPERATIVO

Esta sección es una introducción sencilla al Sistema Operativo (S.O.) de los computadores ATARI. También contiene una breve descripción de los elementos del Sistema Operativo. Ellos se describen en detalle a través de las siguientes secciones y en otras partes de este libro:

8.2 El subsistema de entrada/salida
8.3 Los administradores de interrupción
8.4 Los vectores de las rutinas del sistema
8.5 El monitor
8.6 La base de datos del Sistema
8.7 Los relojes del Sistema y de Eventos
El juego de caracteres en ROM
El paquete de coma flotante

El Sistema Operativo permite al programador de aplicaciones tener acceso a todas las capacidades de los circuitos del computador. Los circuitos del Sistema de Computación Personal ATARI son capaces de proveerlo con algunas funciones de entrada/salida (I/O) excelentes, a través del subsistema de entrada/salida. El subsistema de entrada/salida está constituìdo por un juego de rutinas del sistema, conectado con los circuitos de entrada/salida. El resto del Sistema Operativo soporta a los subsistemas de entrada/salida y da posibilidades adicionales, que pueden usarse en diferentes aplicaciones.

El computador genera interrupciones por varias razones diferentes. Algunas de las interrupciones más comunes son teclado, BREAK, bus serial y borrado vertical.

Los vectores del sistema constituyen el adhesivo que lo mantiene unido. El Sistema Operativo usa los vectores para transladarse de un ambiente de ejecución (BASIC, DUP, Star Raiders[TM]) a otro. Puede llamarse cualquier rutina del sistema, saltando a su vector. Los vectores se usan con mayor frecuencia para llamar las rutinas del sistema de entrada/salida, ajustar los relojes y transferir el control a los diferenters ambientes de ejecución.

Las rutinas del sistema pueden vectorizarse de dos formas. Los vectores ROM son ubicaciones que contienen instrucciones de salto JMP a rutinas del sistema y que no pueden alterarse. Los vectores RAM son ubicaciones que contienen direcciones de rutinas del sistema alterables. Se garantiza que se mantienen las ubicaciones de ambos tipos de

vectores en futuras ediciones del Sistema Operativo.

Sin embargo, debe destacarse que el Sistema Operativo de los modelos XL difiere en algunos aspectos y también en alguno de estos vectores con respecto al Sistema Operativo de los modelos 400 y 800. Mediante el disco "The Translator", es posible simular en gran medida el sistema operativo de los 400/800 en los XL. De hecho, el "translator" contiene el sistema operativo 400/800 y lo carga en el área de RAM subyacente a la ROM del sistema operativo XL. Naturalmente no es posible simular por esta vía las diferencias circuitales entre ambas familias, com son la diferente cantidad de puertas de juego, la tecla (HELP), etc., por lo que nunca podrá conseguirse una simulación perfecta.

El monitor del S.O. es una rutina del sistema, que inicializa el computador al energizarlo y con SYSTEM RESET. El monitor inicializa el subsistema de entrada/salida, establece los vectores y escoje el ambiente de ejecución, una vez que se haya completado la inicialización.

El Sistema Operativo es soportado por una gran base de datos, que consiste de varias banderas de sistema, memorias de entrada/salida y registros de pantalla y gráfica. La mayor parte de la base de datos está dedicada al subsistema de entrada/salida. La base de datos también tiene ubicaciones usadas por otras partes del Sistema Operativo. Los programadores de aplicación pueden aprovechar esta base de datos y agregar cualidades a sus programas.

El Sistema Operativo tiene dos tipos de relojes, relojes de sistema y relojes circuitales. Los relojes de sistema son usados por los programas de aplicación, como relojes programables de uso general. Los relojes circuitales se usan para dar los compases a acontecimientos en tiempo real, tales como las líneas de barrido de televisión en un despliegue de pantalla.

El juego de caracteres en ROM es usado por los administradores de despliegue, para escribir caracteres sobre la pantalla del televisor. Si Ud. quiere, puede crear su propio juego de caracteres e indicarle al Sistema Operativo que lo use en vez del primitivo. Los detalles se dan en el capítulo 3.

El paquete de coma flotante está constituido por un conjunto de rutinas matemáticas que están disponibles al usuario. Las rutinas usan aritmética de decimal codificado en binario (BCD). Se dan las rutinas para ejecutar las funciones aritméticas normales (+, -, *, /) como también las conversiones de ATASCII a BCD y de BCD a ATASCII. Hay una descripción más amplia del paquete de coma flotante, como también la forma de usarlo, en la sección 8 del manual del usuario del Sistema Operativo. El apéndice V de este libro contiene un ejemplo que hace uso de este paquete.

ESTRUCTURA DEL SUBSISTEMA ENTRADA/SALIDA (I/O)

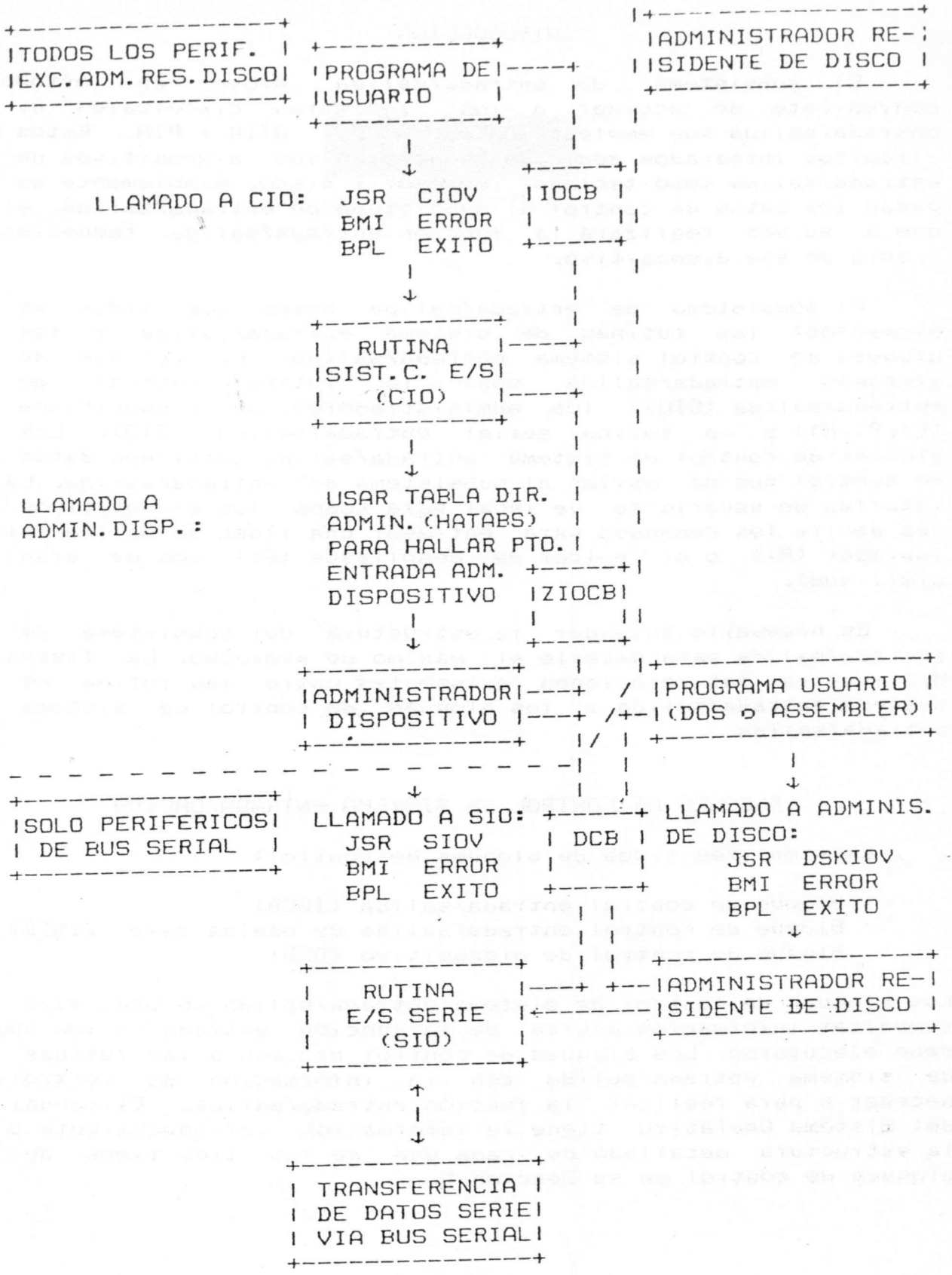


FIGURA 8.1 SUBSISTEMA DE ENTRADA/SALIDA (I/O)

SECCION 8.2 - - SUBSISTEMA DE ENTRADA/SALIDA (I/O)

INTRODUCCION

El subsistema de entrada/salida provee un método conveniente de acceder a los registros circuitales de entrada/salida que emplean ANTIC, POKEY, GTIA y PIA. Estos circuitos integrados dedicados controlan los dispositivos de entrada/salida como teclado, impresor y disco. Simplemente se pasan los datos de control al subsistema de entrada/salida, él que a su vez realizará la función entrada/salida requerida, propia de ese dispositivo.

El subsistema de entrada/salida posee dos tipos de elementos: las rutinas de sistema entrada/salida y los bloques de control sistema entrada/salida. Las rutinas de sistemas entrada/salida son: la rutina central de entrada/salida (CIO), los administradores de dispositivos (E:, P:, K:) y la rutina serial entrada/salida (SIO). Los bloques de control de sistema entrada/salida contienen datos de control que se envìan al subsistema de entrada/salida. La interfaz de usuario se ve igual para todos los dispositivos (es decir, los comandos para entregar una lìnea de salida al impresor (P:) o al editor de despliegue (E:) son de gran similitud).

Es necesario entender la estructura del subsistema de entrada/salida para sacarle el máximo de provecho. La figura 8.1 muestra las relaciones existentes entre las rutina de sistema entrada/salida y los bloques de control de sistema entrada/salida.

BLOQUES DE CONTROL DE SISTEMA ENTRADA/SALIDA

Existen tres tipos de bloques de control:

Bloque de control entrada/salida (IOCB)
Bloque de control entrada/salida de página cèro (ZIOCB)
Bloque de control de dispositivo (DCB)

Los bloques de control de sistema entrada/salida se usan para comunicar información acerca de la función entrada/salida que debe ejecutarse. Los bloques de control proveen a las rutinas de sistema entrada/salida con la información de control necesaria para realizar la función entrada/salida. El manual del Sistema Operativo tiene la información correspondiente a la estructura detallada de cada uno de los tres tipos de bloques de control en su Sección 5.

Tabla del Bloque de Control de Dispositivo (DCB)

FUNCION	NOMBRE	UBICACION	UNIDADES DE DISCO 810/815						ESCRIBIR
			LEER SECTOR		ESCRIBIR SECTOR		PONER	FORMATEAR	IMPR.820
			810	815	810	815			
Id.Bus Serial	DDEVIC	[$0300]	$30	$30	$30	$30	$30	$30	$40
Número Disposit.	DUNIT	[$0301]	1-4	1-8	1-4	1-8	1-4	1-4	1
Byte de Comando	DCOMND	[$0302]	$52	$52	$57	$57	$50	$21	$57
Status	DSTATS	[$0303]	$40	$40	$80	$80	$80	$40	$8[illegible]
Dir.Mem.Transfer.	DBUFLO	[$0304]	U	U	U	U	U	U	U
	DBUFHI	[$0305]	U	U	U	U	U	U	U
Límite de Tiempo	DTIMLO	[$0306]	$30	$30	$30	$30	$31	$130	5
Largo Mem.Transf.	DBYTLO	[$0308]	$80	00	$80	00	$80/00	–	$40
	DBYTHI	[$0309]	$00	01	$80	01	$00/01	–	$00
	DAUX1	[$030A]	2*	2*	2*	2*	- 2*	–	1*
	DAUX2	[$030B]	2*	2*	2*	2*	- 2*	–	1*

FIGURA 8.3

1* = Este byte determina el modo del impresor (vea el manual del modelo 820).
2* = DAUX1+DAUX2 especifican el sector para LEER, ESCRIBIR (PONER), o ESCRIBIR c. verific.
U = indica dirección dada por usuario.
– = indica que se ignora.

Tabla de los bloques de control de Entrada/Salida (IOCB)

LLAMADO	ICHID	ICDNO	ICCOM	ICSTA	ICBAL	ICBAH	ICPTL	ICPTH	ICBLL	ICBLH	ICAX1	ICAX2
ABRIR ARCH.-LECT.	X	X	3	nota1	$80	06	X	X	X	X	4	0
ABRIR ARCH.-ESCR.	X	X	3	nota1	$80	06	X	X	X	X	8	nota2
RECIBIR(GET) BYTE	X	X	7	nota1	00	06	X	X	$80	00	X	X
PONER(PUT) BYTE	X	X	$B	nota1	00	06	X	X	$80	00	X	X
RECIBIR(GET) REG.	X	X	5	nota1	00	06	X	X	$80	00	X	X
PONER(PUT) REG.	X	X	9	nota1	00	06	X	X	$80	00	X	X
CERRAR ARCHIVO	X	X	$C	nota1	X	X	X	X	X	X	X	X
STATUS	X	X	$D	nota1	X	X	X	X	X	X	X	X

FIGURA 8.2

NOTA 1 = El status de un comando de entrada/salida (I/O) se almacena tanto aquí, como en el REG. Y, al retorno desde el CIO.

NOTA 2 = Los bytes auxiliares de los CIO, son empleados por algunos administradores para indicar modos especiales.

X = Indica que puede ignorarse el valor, pero éste no debe ser alterado.

NOTA GENERAL: Las precedentes definiciones de los IOCB suponen:

```
*=$0600
IOBUFF   .RES    80              MEM.TRANSF.I/O USUARIO
ARCH     .BYTE   'D:MIPROG.BAS   NOMBRE ARCHIVO USUARIO
```

Los ocho IOCB del Sistema Operativo se usan para la comunicación entre los programas de usuario y el CIO. La Figura 8.2 muestra el contenido de un IOCB para algunas de las funciones de entrada/salida más corrientes. Los IOCB son:

Nombre	Ubicación, largo
IOCB0	[$340,16]
IOCB1	[$350,16]
IOCB2	[$360,16]
IOCB3	[$370,16]
IOCB4	[$380,16]
IOCB4	[$390,16]
IOCB5	[$3A0,16]
IOCB6	[$3B0,16]
IOCB7	[$3C0,16]

El segundo tipo de bloque de control, ZIOCB [$0020,16], se emplea para comunicar datos de control entrada/salida entre el CIO y los administradores de dispositivo. Al ser llamado, el CIO usa el valor contenido en el registro X como índice que apunta a la dirección de partida del IOCB (uno de los 8), que debe emplearse. El CIO a continuación mueve los datos de control desde el IOCB seleccionado hasta el ZIOCB para ser usados por el administrador de dispositivo apropiado. El ZIOCB es de poco interés salvo que uno quiera escribir su propio administrador de dispositivo o desee reemplazar alguno existente. Vea en la Sección 9 del manual de usuario del Sistema Operativo para más información respecto al ZIOCB.

Los administradores de dispositivo a su vez cargan la información de control en el DCB [$0300,16]. SIO usará la información del DCB y retornará la información de status al DCB, para su posterior uso por parte del administrador del dispositivo. Solamente los administradores de dispositivo que emplean el bus serial hacen uso de DCB y de SIO. La Sección 9 del manual de usuario del Sistema Operativo contiene una descripción detallada del DCB. La Figura 8.3 ilustra algunas funciones comunes de entrada/salida y los contenidos de sus DCB asociados.

El administrador residente de disco no se comporta de acuerdo a la secuencia de llamado usuario - CIO - administrador - SIO regular. En vez de ello, se usa el DCB para comunicar directamente con el administrador residente de disco. El Capìtulo 9 de este libro contiene más información sobre el administrador residente de disco.

RUTINA CENTRAL DEL SISTEMA ENTRADA/SALIDA - CIO

La función principal del CIO es encaminar los datos de control entrada/salida hacia el administrador de dispositivo correcto y a continuación entregar el control a ese administrador. El CIO también actúa como rutina de salida común para todos los administradores de dispositivos. El CIO es un punto de entrada común para la mayoría de las funciones de entrada/salida del Sistema Operativo. Por ejemplo, BASIC llamará el CIO, al ejecutar una sentencia OPEN. El CIO soporta las siguientes funciones:

OPEN	Abrir dispositivo/archivo
CLOSE	Cerrar dispositivo/archivo
GET CHARS	Leer de uno a N caracteres
READ RECORD	Leer el siguiente registro
PUT CHARS	Escribir 1 a N caracteres
WRITE RECORD	Escribir el siguiente registro
STATUS	Recibir el estado del dispositivo
SPECIAL	Específico del administrador de dispositivo (p.ej.: NOTE para el FMS)

Es posible que Ud. quiera hacer sus propios llamados al CIO. La secuencia de llamado para el CIO es:

```
                    ;rem el usuario ya preparó el IOCB
LDX  #IOCBNUM       ;poner índice del IOCB (IOCB*16)
JSR  CIOV           ;vector de rutina del sistema al CIO
BMI  ERROR          ;con desvío, CIO retorna un código de
                    ;error en el registro Y
```

Como se muestra en el llamado anterior, se usa uno de los IOCB para entregar datos de control al CIO. Puede usarse cualquiera de los 8 IOCB. El CIO espera que el índice del IOCB se encuentre en el registro X. En el retorno, los bits del status del 6502 se ponen para indicar el éxito o el error de la operación de entrada/salida. Si el bit N no está puesto (B'0'), la operación de entrada/salida se realizó exitosamente y el registro Y contendrá un 1. Si el bit N está puesto (B'1'), el requerimiento de entrada/salida arrojá un error; el registro Y contendrá el código de error correspondiente. El valor de error/éxito también se retorna en el byte ICSTA (vea la definición del IOCB) del IOCB. El Capítulo 5 del manual de usuario del Sistema Operativo contiene un segmento de programa ejemplo, que llama al CIO para abrir (OPEN) un archivo de disco, leer (READ) algunos registros y cerrar (CLOSE) el archivo.

El CIO encamina los datos de control entrada/salida, usando el índice del IOCB, que se encuentra en el registro X, para mover el contenido del IOCB hasta el ZIOCB. El CIO a

continuación calcula el punto de entrada del administrador del dispositivo y vectoriza hacia la rutina del administrador de dispositivo que corresponda. El Apendice VI es un diagrama de flujo de la rutina del Sistema CIO.

El CIO calcula el punto de entrada del administrador de dispositivo en forma indirecta. Duante un llamado OPEN, el CIO recibe la especificación del dispositivo del archivo, que debe abrirse. Supongamos que el dispositivo debe abrir (OPEN) al impresor; la especificación de dispositivo para el impresor será 'P:'.

El CIO usa una tabla llamada HATABS (figura 8.4A) para calcular indirectamente el punto de entrada al administrador. Esta tabla usa una especificación de dispositivo como clave para encontrar la dirección del punto de entrada del administrador correspondiente. La especificación de dispositivo se usa para encontrar la entrada a la tabla HATABS para el dispositivo que se quiera abrir (OPEN). El CIO comienza en el extremo final de HATABS y busca la primera entrada que calza con la especificación del dispositivo que está buscando, (en nuestro caso 'P:', la primera entrada en la tabla HATABS). La dirección asociada con la especificación del dispositivo constituye un puntero a la lista de los puntos de entrada de administradores (los puntos de entrada de administrador para el impresor se indican en la Figura 8.4B).

El CIO recurre a ICCOM, el byte de comando del IOCB, para encontrar a cual de los puntos de entrada del administrador debe vectorizar. Las tablas de puntos de entrada para todos los administradores de dispositivo residentes se encuentran en el listado del Sistema Operativo . Las respectivas posiciones de todas las tablas de puntos de entrada de administradores de dispositivo tienen el mismo significado. Por ejemplo, la primera posición de todas las tablas de puntos de entrada de administrador de dispositivo es el vector a la rutina de apertura (OPEN) del administrador del dispositivo.

```
                01 ;  TABLA DE DIRECCIONES ADMINISTRADORES
E430            02 PRINTV =             $E430
E440            03 CASETV =             $E440
E400            04 EDITRV =             $E400
E410            05 SCRENV =             $E410
E420            06 KEYBDV =             $E420
                07 ;
0000            08          *=          $031A
                09 ;
                10 HATABS
031A            20          .BYTE       'P'     IMPRESOR
031B 30E4       30          .WORD       PRINTV    TABLA PUNTO ENTRADA
031D 43         40          .BYTE       'C'     CASSETTE
031E 40E4       50          .WORD       CASETV    TABLA PUNTO ENTRADA
0320 45         60          .BYTE       'E'     EDITOR DESPLIEGUE
0321 00E4       70          .WORD       EDITV     TABLA PUNTO ENTRADA
0323            80          .BYTE       'S'     ADMINISTRADOR PANTALLA
0324 10E4       90          .WORD       SCRENV    TABLA PUNTO ENTRADA
0326 4B         0100        .BYTE       'K'     TECLADO
0327 20E4       0110        .WORD       KEYBDV    TABLA PUNTO ENTRADA
0329 00         0120        .BYTE       0       ENTRADA LIBRE 1 (DOS)
032A 00 00      0130        .BYTE       0,0
032C 00         0140        .BYTE       0       ENTRADA LIBRE 2 (850)
032D 00 00      0150        .BYTE       0,0
032F 00         0160        .BYTE       0       ENTRADA LIBRE 3
0330 00 00      0170        .BYTE       0,0
0332 00         0180        .BYTE       0       ENTRADA LIBRE 4
0333 00 00      0180        .BYTE       0,0
0335 00         0200        .BYTE       0       ENTRADA LIBRE 5
0336 00 00      0210        .BYTE       0,0
0338 00         0220        .BYTE       0       ENTRADA LIBRE 6
0339 00 00      0230        .BYTE       0,0
033B 00         0240        .BYTE       0       ENTRADA LIBRE 7
```

FIGURA 8.4A TABLA DE DIRECCIONES DE ADMINISTRADORES (HATABS)

```
        *=$PRINTV
E430 9E EE      .WORD   PHOPEN-1    ABRIR (OPEN) DISPOSITIVO
E432 DB EE      .WORD   PHCLOS-1    CERRAR (CLOSE) DISPOSITIVO
E434 9D EE      .WORD   BADST-1     LEER (READ) DISP.-NO IMPLEM.
E436 A6 EE      .WORD   PHWRIT-1    ESCRIBIR (WRITE) DISPOSITIVO
E438 80 EE      .WORD   PHSTAT-1    STATUS DISPOSITIVO
E43A 9D EE      .WORD   BADST-1     ESPECIAL -NO IMPLEMENTADO
E43C 4C 78 EE   JMP     PHINIT      INICIALIZACION DISPOSITIVO
```

FIGURA 8.4B TABLA DE PUNTO DE ENTRADA DEL IMPRESOR

HATABS se encuentra en RAM en la dirección $031A y contiene lugar para 14 ingresos. Al energizar y con SYSTEM RESET los contenidos de HATABS se inicializan tal como se indica en la Figura 8.4. En ese momento HATABS tiene ingresos para todos los administradores residentes, excepto el del disco. El administrador del disco nunca se llama a través de HATABS (Vea el Capítulo 9).

El usuario o el Sistema Operativo pueden agregar otros ingresos a HATABS. El Sistema Operativo puede agregar ingresos como parte de su función de energización o SYSTEM RESET. Cualquier nueva entrada partiría en $0329. Posibles agregados a HATABS serían ingresos para la unidad de disco (el administrador de archivos o FMS), el módulo de interfaz 850 o el modem 1030.

Puede sacarse ventaja de la naturaleza flexible de HATABS, para agregar algunas nuevas posibilidades al Sistema Operativo. Un ejemplo (Figura 8.5) indica como agregar un administrador nulo. Un administrador nulo es exactamente eso, no realiza ninguna función. Puede usarse por ejemplo, para encontrar defectos en un programa que envuelve dispositivos de acceso lento. En vez de esperar por 50.000 accesos de disco para encontrar un defecto, simplemente haga pasar la salida a través del administrador nulo.

Otra aplicación de HATABS es cambiar la función de una antigua entrada en ella. Suponga que Ud. quiere agregar un impresor a su computador que tenga alguna función especial, que no es soportada por el administrador de impresor normal. Cambiando el puntero al punto de entrada de la tabla HATABS puede apuntar a toda entrada/salida 'P:' a su propio administrador de impresor. El Apendice VII muestra un ejemplo de administrador de impresor Qume que hace uso de las puertas de controladores para la rápida transferencia de datos, para impresión.

```
0000            10      *= $0600
   =031A        20 HATABS = $031A
0600            30 START
0600 A000       40      LDY #0
0602            50 BUCLE
0602 B91A03     60      LDA HATABS,Y
0605 C900       70      CMP #0          ENTRADA DISPONIBLE?
0607 F009       80      BEQ AQUI
0609 C8         90      INY
060A C8         0100    INY
060B C8         0110    INY             APUNTAR A LA PROXIMA ENTRADA DE HATABS
060C C022       0120    CPY #34         AL FINAL DE HATABS?
060E D0F2       0130    BNE BUCLE       NO...
0610 38         0140    SEC             SI, INDICAR ERROR AL MANDANTE
0611 60         0150    RTS
                0160 ;
0612            0170 AQUI
0612 A94E       0180    LDA #'N         NOMBRE DEL DISPOSITIVO
0614 991A03     0190    STA HATABS,Y
0617 C8         0200    INY
0618 A924       0210    LDA #TABNULO&255
061A 991A03     0220    STA HATABS,Y    TABLA ENTRADA ADMINISTRADOR NULO
061D C8         0230    INY
061E A906       0240    LDA #TABNULO/256
0620 991A03     0250    STA HATABS,Y
0623 60         0260    RTS
                0270 ;
0624            0280 TABNULO
0624 3206       0290    .WORD ADMIN-1   OPEN
0626 3206       0300    .WORD ADMIN-1   CLOSE
0628 3406       0310    .WORD NADA-1    READ
062A 3206       0320    .WORD ADMIN-1   WRITE
062C 3206       0330    .WORD ADMIN-1   STATUS
062E 3406       0340    .WORD NADA-1    ESPECIAL
0630 4C3306     0350    JMP ADMIN       INICIALIZACION
                0360 ;
0633            0370 ADMIN
0633 A001       0380    LDY #1          FUNCION I/O EXITOSA
                0390 NADA               FUNCION  NO IMPLEMENTADA
0635 60         0400    RTS
```

FIGURA 8.5 ADMINISTRADOR NULO

ACCEDIENDO EL CIO DESDE BASIC

La mayoría de las funciones CIO (OPEN, CLOSE, etc.) están accesibles a través de llamadas desde BASIC. BASIC carece de un juego de funciones del CIO: la posibilidad de hacer entrada/salida de algo que no sea un registro y que tenga más de un byte, de una vez (GETCHRS y PUTCHRS).

La posibilidad de ingresar o sacar una memoria de transferencia de caracteres, todos de una vez, es una posibilidad muy atractiva. Por ejemplo, una rutina en lenguaje assembler puede cargarse directamente en la memoria, desde un archivo de disco. En un programa BASIC, una rutina en lenguaje assembler normalmente se lee como string y se hace un llamado USR a la dirección del string (ADR(string)). Como en BASIC la dirección de un string puede cambiar a raíz de una modificación del programa, la rutina de lenguaje assembler debe ser independiente de su ubicación. Esto significa que, instrucciones con referencias de memoria a direcciones al interior del string, no funcionarán.

La subrutina de Figura 8.6 evita el uso de strings. En esta forma no es necesario que el módulo assembler sea independiente de su ubicación. Los datos de control se ingresan por POKE en el IOCB que leerá la rutina de lenguaje assembler directamente a la RAM en la dirección en la cual fue ensamblada. La subrutina BASIC de la Figura 8.6 también puede usarse para entregar directamente datos a la salida desde la memoria, siempre que el usuario especifique tanto la ubicación como el largo de la memoria de transferencia de datos.

```
30 REM ESTE PROGRAMA CARGA LA PAG.6 DESDE EL ARCH. D:PRUEBA
100 DIM ARCH$(20),CIO$(7):CIO$="hhh\*\LV\d\"
102 REM \ \ indica video inverso.
105 REM CIO$ es PLA,PLA,PLA,TAX,JMP $E456 (CIOV)
110 ARCH$="D:PRUEBA":REM *NOMBRE ARCHIVO
120 CMD=7:DIRIN=1536:GOSUB 30000
130 IF ERROR()1 THEN ? "ERROR - ";ERROR;" EN LINEA ";
135 ? PEEK(186)+256*PEEK(187),PEEK(195)
200 END
300 REM -      RUTINA DE DISPOSICION DEL CIO
310 REM
30000 REM RUTINA ORIGINAL POR E. EKBERG PARA ATARI 03-09-80
30001 REM
30002 REM CARGA O GRABA UN ARCHIVO BINARIO DESDE BASIC
30003 REM DISPONIENDO UN IOCB Y LLAMANDO EL CIO DIRECTAMENTE
30004 REM
30006 REM AL INGRESAR: CMD=7 SIGNIFICA CARGAR MEMORIA
30008 REM _            CMD=11 SIGNIFICA GRABAR MEMORIA
30009 REM _            DIRIN= DIRECCION DE CARGA O DE GRAB.
30010 REM _            BYTES= No. DE BYTES A CARGAR O GRABAR
30011 REM _            IOCB= EL IOCB A USAR
30012 REM _            ARCH$=NOMBRE DEL ARCHIVO DESTINO
30013 REM _
30014 REM AL EGRESAR: ERROR=1 SIGNIFICA COMANDO EXITOSO
30018 REM _           ERROR()1 INDICA EL STATUS DEL ERROR
30019 REM
30020 REM _  *** VALORES DEL IOCB ***
30022 REM
30024 IOCBX=IOCB*16:ICCOM=834+IOCBX:ICSTA=835+IOCBX
30026 ICBAL=836+IOCBX:ICBAH=837+IOCBX
30028 ICBLL=840+IOCBX:ICBLH=841+IOCBX
30029 REM
30030 AUX1=4:IF CMD=11 THEN AUX1=8
30035 TRAP 30900:OPEN #IOCB,AUX1,0,ARCH$:IOCBX=IOCB*16
30040 TEMP=DIRIN:GOSUB 30500
30090 POKE ICBAL,LSB:POKE ICBAH,MSB
30100 TEMP=BYTES:GOSUB 30500
30130 POKE ICBLL,LSB:POKE ICBLH,MSB
30140 POKE ICCOM,CMD:ERROR=USR(ADR(CIO$),IOCBX)
30150 ERROR=PEEK(ICSAT):RETURN
30200 REM
30300 REM _  *** RUTINA ENTREGA MSB,LSB DE NUMEROS DE 16 BITS
30400 REM
30500 MSB=INT(TEMP/256):LSB=INT(TEMP-256*MSB):RETURN
30550 REM
30600 REM ***SI HAY UN ERROR EN LA RUTINA, SE ATRAPA AQUI***
30900 ERROR=PEEK(195)
30920 CLOSE #IOCB:RETURN
```

FIGURA 8.6 LLAMADO CIO DIRECTO DESDE BASIC

LOS ADMINISTRADORES DE DISPOSITIVO

Los administradores de dispositivo pueden dividirse en administradores residentes y no residentes. Los administradores residentes se encuentran en la ROM del Sistema Operativo. Todo administrador residente puede llamarse a través del CIO, siempre que el administrador tenga una entrada en HATABS. Los administradores de dispositivo residentes son:

(E:) EDITOR DE DESPLIEGUE
(S:) PANTALLA (SCREEN)
(K:) TECLADO (KEYBOARD)
(P:) IMPRESOR (PRINTER)
(C:) CASSETTE

Los administradores no residentes no se encuentran en la ROM del Sistema Operativo. Pueden agregarse administradores no residentes al Sistema Operativo durante la energización o con RESET, o también lo puede hacer el usuario durante la ejecución de su programa. Refiérase a la Figura 8.5 como ejemplo de como agregar un administrador al Sistema Operativo.

Los administradores de dispositivo hacen uso de los datos de control entrada/salida que pasa el CIO al ZIOCB. Los datos contenidos en el ZIOCB se usan para realizar funciones de entrada/salida tales como OPEN, CLOSE, PUT y GET. No todos los administradores de dispositivo soportan todos los comandos de entrada/salida (p.e. tratar de hacer PUT de un caracter al teclado dará como resultado un error 146, función no implementada). La Sección 5 del manual del Sistema Operativo contiene una lista de las funciones soportadas por cada uno de los administradores.

RUTINA DEL SISTEMA SERIAL DE ENTRADA/SALIDA - - SIO

EL SIO Y LOS ADMINISTRADORES DE DISPOSITIVOS

El SIO administra los dispositivos a través del sus serial, entre los administradores de dispositivo serie dentro del computador y los dispositivos conectados al bus serial. El SIO se comunica con su mandante a través del bloque de control de dispositivo (DCB). El SIO usa los datos de control entrada/salida, que se encuentran en el DCB, para enviar y recibir comandos y datos a trav+s del bus serial. La secuencia de llamado es la siguiente:

```
                ; el mandante ha preparado el DCB para la función
JSR  SIOV       ; vector del sistema SIO
BMI  ERROR      ; bit N puesto indica error en la ejecución I/O
```

El DCB contiene información de control entrada/salida para SIO y por lo tanto debe prepararse antes del llamado al SIO. La figura 8.3 muestra el contenido del DCB para algunas operaciones de entrada/salida comunes.

Es necesario entender la estructura del DCB para enviar comandos al SIO. El DCB se describe en la sección 9 del Manual del Sistema Operativo. La figura 8.7 muestra una rutina de lenguaje assembler muy simple para enviar una línea al impresor, preparando el DCB y llamando al SIO.

```
0300            05      *= $3000  INICIO ARBITRARIO
                10 ;ESTA RUTINA SACA UNA LINEA AL IMPRESOR LLAMANDO EL SIO
    =E459       20 SIOV =  $E459  VECTOR SIOV
    =009B       30 CR   =  $9B    EOL
    =0040       40 PRNTID =  $40    ID BUS SERIAL IMPRESOR
    =004E       45 MODO =  $4E    MODO NORMAL
    =001C       50 PTIMOT =  $1C    UBICACION LIMITE DE TIEMPO
    =0300       60 DDEVIC =  $0300  ID BUS SERIAL DISPOSITIVO
    =0301       70 DUNIT =  $0301   NUMERO UNIDAD SERIAL
    =0302       80 DCOMND =  $0302  COMANDO SIO
    =0303       90 DSTATS =  $0303  DIRECCION DATOS SIO
    =0304      0100 DBUFLO = $0304  DIRECCION BUFFER LSB
    =0305      0110 DBUFHI = $0305  DIRECCION BUFFER MSB
    =0306      0120 DTIMLO = $0306  LIMITE DE TIEMPO SIO
    =0307      0130 DTIMHI = $0307
    =0308      0140 DBYTLO = $0308  LARGO BUFFER
    =0309      0150 DBYTHI = $0309
    =030A      0160 DAUX1 = $030A   BYTE AUXILIAR---MODO DEL IMPRESOR
    =030B      0170 DAUX2 = $030B   BYTE AUXILIAR---SIN USO
               0180 ;
3000 454A454D  0190 MENS .BYTE "EJEMPLO 12",CR
3004 504C4F20
3008 31329B
               0200 ;
300B A940      0220     LDA #PRNTID DISPONER ID BUS
300D 8D0003    0230     STA DDEVIC
3010 A901      0240     LDA #1      DISPONER NUMERO UNIDAD
3012 8D0103    0250     STA DUNIT
3015 A94E      0260     LDA #MODO
3017 8D0A03    0270     STA DAUX1   MODO DE IMPRESOR NORMAL
301A A901      0275     LDA #1
301C 8D0B03    0280     STA DAUX2   SIN USO
301F 8D0703    0290     STA DTIMHI  TIEMPO LIMITE<256 SEG
3022 A51C      0300     LDA PTIMOT  DISPONER TIEMPO LIMITE SIO PARA IMPRESOR
3024 8D0603    0310     STA DTIMLO
3027 A900      0320     LDA #MENS&255
3029 8D0403    0330     STA DBUFLO  DISPONER MENSAJE COMO BUFFER
302C A930      0340     LDA #MENS/256
302E 8D0503    0350     STA DBUFHI
3031 A980      0360     LDA #$80    DISPONER DIRECCION DATOS SIO PARA RECEPCION EN EL PERIFERICO
3033 8D0303    0370     STA DSTATS
3036 A957      0380     LDA #'W     COMANDO ESCRITURA SIO
3038 8D0203    0390     STA DCOMND
303B 2059E4    0410     JSR SIOV    LLAMAR SIO
303E 3001      0420     BMI ERROR
3040 00        0430 BIEN BRK
3041 00        0440 ERROR BRK
```

FIGURA 8.7 LLAMADO AL SIO PARA SACAR UNA LINEA AL IMPRESOR

INTERRUPCIONES DEL SIO

El SIO recurre a tres interrupciones IRQ para enviar y recibir comunicaciones de bus serial desde y hacia los dispositivos de bus serial. Ellas son:

IRQ	Ubicación, largo	Función
VSERIR	[$020A,2]	ENTRADA SERIAL LISTA
VSEROR	[$020C,2]	SE REQUIERE SALIDA SERIAL (SALIDA SERIAL LISTA)
VSEROC	[$020E,2]	TRANSMISION COMPLETA

Cuando el SIO usa el bus serial para sus comunicaciones, se detiene toda ejecución de programa. Para las salidas, el SIO pasa al registro de desplazamiento de la salida serie de POKEY (SEROUT) un byte para ser enviado a través del bus serial, enseguida entra a un bucle de espera hasta que reciba un IRQ (VSEROR) de POKEY indicándole a SIO que SEROUT está disponible para el próximo byte. El IRQ provoca un salto a la rutina SIO para cargar SEROUT con el byte siguiente. Este bucle se ejecuta hasta que todos los bytes especificados por el largo de la memoria de transferencia de DCB hayan sido enviados. VSEROR es un IRQ que indica que la transmisión del byte a través del bus se ha completado.

La ejecución de SIO para una entrada es similar. POKEY informa a SIO, que ha recibido un byte en el registro de desplazamiento de ingreso serial (SERIN), generando un IRQ (VSERIR). El SIO almacena el byte en una memoria de transferencia y enseguida SIO se mantiene en el bucle de espera, hasta que el próximo IRQ indica que se ha recibido un nuevo byte.

Ud. puede haber deducido de la explicación anterior que el SIO pierde algún tiempo mientras espera que POKEY envíe o reciba la información del bucle. Como los vectores de las tres rutinas de servicio de IRQ del SIO son vectores RAM, pueden ser aprovechados algunos de los administradores para mejorar el desempeño del sistema de entrada/salida.

De hecho es así como el módulo de interfaz 850 es capaz de hacer él la operación de entrada/salida (I/O) concurrente. El administrador del módulo 850 se hace cargo de los vectores IRQ del SIO y apunta estos vectores IRQ a sus propias rutinas de IRQ, mientras se encuentre en el modo I/O concurrente . Con esto el administrador del módulo 850 puede enviar comandos a través del bus. Mientras espera que el módulo 850 realice el comando, el administrador del módulo 850 permite que el programa continúe su desarrollo.

SECCION 8.3 - - INTERRUPCIONES

INTRODUCCION

En la sección previa hemos visto como el SIO recurre a interrupciones para coordinar la transferencia de datos a través del bus serial. El computador también tiene otras categorías de interrupciones. Pueden usarse varias de estas otras interrupciones para agregar algunas características muy poderosas a aplicaciones específicas. Debe aclararse que existen dos tipos de interrupciones, las enmascarables (IRQ) y las no enmascarables (NMI). Las interrupciones del sistema son:

Nombre (vector)	Tipo	Función	Usado por
LISTA DE DESPLIEGUE (VDSLST)	NMI	sincronización gráfica	Usuario
RESET (ninguno)	NMI	inicialización sistema	Sistema
BORRADO VERTICAL (VVBLKI,VVBLKD)	NMI	Despliegue gráfico	Sistema, usuario
SALIDA SERIAL LISTA (VSERIN)	IRQ	ENTRADA SERIAL	Sistema
ENTRADA SERIAL LISTA (VSEROR)	IRQ	Entrada serial	Sistema
SALIDA SERIAL COMPLETA (VSEROC)	IRQ	Salida serial	Sistema
RELOJ 1 POKEY (VTIMR1)	IRQ	Reloj circuital	Usuario
RELOJ 2 POKEY (VTIMR2)	IRQ	Reloj circuital	Usuario
*RELOK 4 POKEY (VTIMR4)	IRQ	Reloj circuital	Usuario
TECLADO (VKEYBD)	IRQ	Presión de tecla	Sistema
TECLA BREAK (ninguno)	IRQ	Presión de BREAK	Sistema
ACCION BUS SERIAL (VPRCED)	IRQ	acción del dispositivo	Sin uso
INTERRUPCION BUS SERIAL (VINTER)	IRQ	Interrupción dispositivo	Sin uso

* Este IRQ no se encuentra vectorizado en el Sistema Operativo actual.

Si Ud. no se ha familiarizado con las interruciones, vea la sección 6 del manual del Sistema Operativo, que contiene información acerca de ellas. Trabajar con interrupciones puede ser intrincado. Por ejemplo, si accidentalmente se inhibe la interrupción del teclado, el computador ignorará todas las presiones sobre el teclado, excepto la de la tecla BREAK. Aunque esto pueda ser útil en algunos casos,, hará muy difícil la purga de los defectos de un programa.

EL ADMINISTRADOR DE LA INTERRUPCION IRQ

El Sistema Operativo tiene un administrador de interrupciones IRQ, que procesa los divesos IRQ. El administrador IRQ tiene vectores RAM para todos los IRQ, excepto él de la tecla BREAK. Los vectores IRQ se ponen a sus valores iniciales, tanto durante la energización, como el RESET del sistema. La ubicación de los vectores RAM de IRQ se describe en la Figura 8.9B.

Los vectores IRQ son:

VIMIRQ - vector de IRQ inmediato. Todos los IRQ vectorizan a través de esta ubicación. VIMIRQ normalmente apunta al administrador de IRQ. Ud. puede desviar este vector para hacer su propio procesamiento de interrupciones IRQ.

VSEROR - vector IRQ de salida serial de POKEY lista (vea la Sección 8.2).

VSERIN - vector IRQ de entrada serial POKEY lista (vea la Sección 8.2).

VSEROC - vector IRQ de salida serial POKEY completa (vea la Sección 8.2).

VTIMR1 - vector IRQ del reloj 1 de POKEY (vea la Sección 8.7 para más información acerca de los relojes POKEY).

VTIMR2 - vector IRQ del reloj 2 POKEY.

VTIMR4 - vector IRQ del reloj 4 POKEY.

VKEYBD - vector IRQ del teclado. El presionar cualquier tecla, excepto BREAK, produce este IRQ. VKEYBD puede emplearse para preprocesar el código de tecla antes de convertirlo a ATASCII (por parte del Sistema Operativo). VKEYBD normalmente apunta a la rutina IRQ de teclado del Sistema Operativo.

VPRCED - vector IRQ de acción de periférico. La línea de acción está disponible para los periféricos conectados al bus serial. Por ahora este IRQ no tiene uso y normalmente apunta a un RTI.

VINTER - vector IRQ de interrupción de periférico. También esta línea de interrupción está disponible en el bus serial. VINTER también apunta normalmente a un RTI.

VBREAK - vector IRQ de la instrucción BRK del 6502. Cada vez que se ejecute un código operacional $00 (la instrucción software de break), ocurre esta interrupción. VBREAK puede usarse para establecer puntos break en un corrector de

programa (DEBUGGER). VBREAK normalmente apunta a un RTI.

Los IRQ se habilitan y se inhiben como grupo por medio de las instrucciones CLI y SEI del 6502 respectivamente. Los IRQ también tienen bits individuales de habilitación/inhibición. La Sección 3 del manual de circuitos (Haroware Manual) indica los IRQ y sus bits de habilitación/inhibición respectivos.

El registro (IRQEN) contiene la mayor parte de los bits de habilitación/inhibición IRQ y es un registro de escritura solamente. El Sistema Operativo mantiene una copia de IRQEN en POKMSK para el ususrio. POKMSK se transfiere a IRQEN durante el intervalo vertical.

USANDO LOS IRQ

Muchas aplicaciones requieren de un teclado 'a prueba de idiotas'. Esto significa que el usuario puede presionar cualquier tecla o combinación de teclas y que el programa solamente aceptará secuencias de teclado válidas, ignorando las inválidas. Pueden usarse una serie de vectores IRQ para proteger un programa contra idiotas. El ejemplo de Figura 8.8 usa el vector IRQ VKEYD para inhibir la tecla de CONTROL. La rutina también enmascara la tecla BREAK, desviando el vector VIMIRQ e ignorando las interrupciones de la tecla BREAK.

EL ADMINISTRADOR DE NMI

El Sistema Operativo tiene un administrador NMI para manejar las interrupciones no enmascarables. A diferencia de los IRQ, los NMI no pueden ser enmascarados (inhibidos) en el 6502. Todos los NMI, exceptuando SYSTEM RESET, pueden inhibirse en ANTIC.

Dos de los NMI, la interrupción de lista de despliegue (DLI) y la interrupción de intervalo de borrado vertical (VBLANK), tienen vectores RAM que pueden usarse. De hecho, VBLANK puede interrumpirse en dos puntos, VBLANK inmediato o diferido. Los vectores NMI son:

Nombre	Vector
SYSTEM RESET	ninguno
INTERRUPCION DE LISTA DE DESPLIEGUE	VDSLST [$0200]
BORRADO VERTICAL	
INMEDIATO	VVBLKI [$0222]
DIFERIDO	VVBLKD [$0224]

El NMI SYSTEM RESET no tiene vector RAM. SYSTEM RESET siempre produce un salto hacia la rutina de partida en caliente (WARMSTART) del monitor (Seccion 8.5). Las interrupciones DLI y VBLANK se cubren respectivamente en el Capìtulo 5 y el Apéndice I.

```
      =0010      10 POKMSK =  $10
      =D209      20 KBCODE =  $D209
      =0208      30 VKEYBD =  $0208
      =D20E      40 IRQEN =   $D20E
      =D20E      45 IRQST =   IRQEN
      =0216      46 VMIRQ =   $0216
0000             60       *=  $0600
0600 78          80 START SEI         INHIBIR IRQS
0601 AD1602      90       LDA VMIRQ   REEMPLAZO DEL VECTOR IRQ
0604 8D4D06      0100     STA NBRK+1  CON EL PROPIO
0607 AD1702      0110     LDA VMIRQ+1 TODOS LOS IRQ
060A 8D4E06      0120     STA NBRK+2  IRAN A NBRK
060D A945        0130     LDA #IRQ&255
060F 8D1602      0140     STA VMIRQ
0612 A906        0150     LDA #IRQ/256
0614 8D1702      0160     STA VMIRQ+1
0617 58          0170     CLI         HABILITAR IRQS
0618 AD0802      0200     LDA VKEYBD  APUNTAR IRQ DE TECLADO A REP
061B 8D4306      0210     STA SALTO+1 A REP
061E AD0902      0220     LDA VKEYBD+1
0621 8D4406      0230     STA SALTO+2
0624 A939        0240     LDA #REP&255 VECTOR IRQ TECLA
0626 8D0802      0250     STA VKEYBD  LSB DEL VECTOR
0629 A906        0260     LDA #REP/256
062B 8D0902      0270     STA VKEYBD+1
062E             0280 RTS
062E             0290     *=  $0639
0639 AD29D2      0300 REP LDA KBCODE  TODOS LOS IRQ DE TECLAS LLEGAN AQUI
063C 2950        0310     AND #80     VERIFICAR SI CONTROL ESTA PRESIONADO
063E F002        0320     BEQ SALTO   EN CASO NEGATIVO IR
0640 68          0330     PLA         DE OTRA FORMA IGNORAR TECLA CONTROL
0641 40          0340     RTI
0642 4C4206      0360 SALTO JMP SALTO ESTO LLAMA EL ANTIGUO IRQ DE TECLA
0645 48          0375 IRQ PHA         TODOS LOS IRQ LLEGAN AQUI
0646 AD0ED2      0380     LDA IRQST   VERIFICAR (BREAK)
0649 1004        0390     BPL BREAK   SI HAY IRQ (BREAK)
064B 68          0405     PLA         CASO CONTRARIO LLAMAR VECTOR IRQ ANTIGUO
064C 4C4C06      0410 NBRK JMP NBRK   LLAMAR ANTIGUO VECTOR IRQ
064F A97F        0430 BREAK LDA #$7F  AQUI SI HAY (BREAK)
0651 8D0ED2      0440     STA IRQST   NO MOSTRAR (BREAK)
0654 A510        0450     LDA POKMSK
0656 8D0ED2      0460     STA IRQEN
0659 68          0462     PLA
065A 40          0464     RTI         RETORNAR COMO SIN OCURRENCIA DE (BREAK)
065B             2470     *=  $02E2
02E2 0006        0480     .WORD START
```

FIGURA 8.8 PROTEGIENDO EL TECLADO

SECCION 8.4 - LOS VECTORES DEL SISTEMA

El Sistema Operativo tiene dos tipos de vectores, vectores ROM y vectores RAM. Los vectores ROM son ubicaciones que contienen instrucciones JMP hacia las rutinas del sistema. Los vectores RAM contienen direcciones de dos bytes a rutinas del sistema, punteros de entrada de administradores (Vea CIO en la Sección 8.2) o a rutinas de inicialización.

Se garantiza que tanto las direcciones de los vectores ROM como de los vectores RAM se mantendrán sin modificación en nuevas ediciones del Sistema Operativo ATARI; sin embargo, los contenidos de estos vectores pueden cambiar con libertad. Los vectores, su contenido y una breve descripción de su función se dan en las Figuras 8.9A y 8.9B.

VECTORES ROM

Nombre	Ubicación	Uso
DISKIV	$E450	inicialización del administrador de disco
DSKINV	$E453	vector del administrador de disco
CIOV	$E456	vector de la rutina de entrada/salida central
SIOV	$E459	vector de la rutina de entrada/salida serial
SETVBV	$E45C	vector de la rutina de puesta de relojes de sistema
SYSVBV	$E45F	cálculos del borrado vertical del sistema
XITVBV	$E462	salida de los cálculos de borrado vertical
SIOINV	$E465	inicialización de entrada/salida serial
SENDEV	$E468	rutina de habilitación de transmisión de bus serial
INTINV	$E46B	rutina del administrador de interrupción
CIOINV	$E46E	inicialización de entrada/salida central
BLKBDV	$E471	vector del modo pizarra (MEMO PAD)
WARMSV	$E474	punto de entrada a partida en caliente (SYSTEM RESET)
COLDSV	$E477	punto de entrada de partida en frío (energización)
RBLOKV	$E47A	vector de rutina lectura bloque cassette
CSOPIV	$E47D	vector de apertura de cassette para entrada

Un ejemplo de uso de un vector ROM es:

```
JSR CIOV
```

FIGURA 8.9A VECTORES ROM

VECTORES RAM

Nombre	Ubicación	Valor	Uso
VDLST	$0200	$E7B3	Vector NMI int. l. despliegue
VPRCED	$0202	$E7B3	Vector IRQ línea acción (s.u.)
VINYER	$0204	$E7B3	Vector IRQ línea interr. (s.u.)
VBREAK	$0206	$E7B3	Vector IRQ instr. BREAK de softw.
VKEYBD	$0208	$FFBE	Vector IRQ teclado
VSERIN	$020A	$EB11	Vector IRQ entr. serial lista
VSEROR	$020C	$EA90	Vector IRQ sal. serial lista
VSEROC	$020E	$EAD1	Vector IRQ sal. serial completa
VTIMR1	$0210	$E7B3	Vector IRQ reloj POKEY 1
VTIMR2	$0212	$E7B3	Vector IRQ reloj POKEY 2
VTIMR4	$0214	$E7B3	Vector IRQ reloj POKEY 4
VIMIRQ	$0216	$E6F6	Vector IRQ inmed. al admin. IRQ
VVBLKI	$0222	$E7D1	Vector NMI borrado vertical
VVBLKD	$0224	$E93E	Vector diferido borrado vertical
CDTMA1	$0226	$xxxx	Dir. JSR reloj sistema No. 1
CDTMA2	$0228	$xxxx	Dir. JSR reloj sistema No. 2
CASINI	$0002	$xxxx	V. inicializ. prog. cass. autopart.
DOSINI	$000C	$xxxx	Vector inicialización de disco
DOSVEC	$000A	$xxxx	Vector ejecuc. software de disco
RUNVEC	$02E0	$xxxx	Vector ejec. automat. arch. DUP
INIVEC	$02E2	$xxxx	V. inic. arch. DUP ej. autom
HATABS	$031A	'P'	Id. dispositivo impresor
	$031B	$E430	dir. tabla ptos. ent. impresor
	$031D	'C'	Id. dispositivo cassette
	$31E	$E440	dir. tabla ptos. ent. cassette
	$0320	'E'	Id. disp. editor de despliegue
	$0321	$E400	dir. tabla ptos. ent. editor despl.
	$0323	'S'	Id. administrador de pantalla
	$0324	$E410	dir. tabla ptos. ent. adm. pant.
	$0326	'K'	Id. administrador de teclado
	$0327	$E420	dir. tabla ptos. ent. adm. teclado
	$0329	'x'	entrada sin uso No. 1 de HATABS
	$032C	'x'	entrada sin uso No. 2 de HATABS
	$032F	'x'	entrada sin uso No. 3 de HATABS
	$0332	'x'	entrada sin uso No. 4 de HATABS
	$0335	'x'	entrada sin uso No. 5 de HATABS
	$0337	'x'	entrada sin uso No. 6 de HATABS
	$033B	'x'	entrada sin uso No. 7 de HATABS
	$033E	'x'	entrada sin uso No. 8 de HATABS

La 'x' indica que el contenido es variable.
(s.u.) indica que no se usa en la actualidad.

Un ejemplo de uso de un vector RAM es:

```
      JSR  LLAMA
LLAMA JMP  (DOSINI)
```

FIGURA 8.9B VECTORES RAM

SECCION E.5 - EL MONITOR

```
                    10 ;ORIGINAL ESCRITO POR MICHAEL ECKBERG
=0600               30 START =  $0600
=000C               40 DOSINI = $0C
=02E7               50 MEMLO =  $02E7
=3000               60 NUEVAM = $3000   ;ALTERAR ESTE VALOR SEGUN NECESIDAD
                    70 ;ESTA RUTINA RESERVA ESPACIO PARA RUTINAS ASSEMBLER
                    90 ; REPONIENDO EL VECTOR MEMLO. TAMBIEN OPERA COMO
                    0100 ; ARCHIVO AUTORUN.SYS. TAMBIEN REPONE MEMLO CON (RESET).
                    0120 ; MEMLO SE PONE EN EL VALOR DADO A NUEVAM.
                    0130 ;
                    0140 ; ESTA PARTE ES PERMANENTE, ES DECIR NO NECESITA SER RESIDENTE
                    0150 ; SE HA MOVIDO EL VECTOR DE INICIACION DEL DOS,
                    0160 ; ALMACENANDOLO EN LA UBICACION INITDOS+1&2.
                    0180 ; SE INICIALIZA DOS Y DSE INICIALIZA MEMLO
                    0190 ; CON (RESET) SE EJECUTA INITDOS.
0000                0200     *= START
0600                0210 INITDOS
0600 200D06         0220     JSR ANTDOSI ; PREPARAR LISTA INIC.
0603 A900           0230     LDA #NUEVAM&255
0605 8DE702         0240     STA MEMLO
0608 A930           0250     LDA #NUEVAM/256
060A 8DE802         0260     STA MEMLO+1
060D                0270 ANTDOSI
060D 60             0280     RTS
                    0290 ; ESTA PARTE SOLO SE EJECUTA AL ENCENDER Y
                    0300 ; PUEDE ELIMINARSE DESPUES DEL ENCENDIDO INICIAL.
                    0330 ; ESTA RUTINA ALMACENA EL CONTENIDO DE DOSINI
                    0350 ;CON UN JSR EN INITDOS+1, A CONTINUACION REMPLAZA
                    0370 ; DOSINI CON SU PROPIO VALOR, LA UBICACION INITDOS
060E                0390 DIREJE
060E A50C           0400     LDA DOSINI  ; SALVAR DOSINI
0610 8D0106         0410     STA INITDOS+1
0613 A50D           0420     LDA DOSINI+1
0615 8D0206         0430     STA INITDOS+2
0618 A500           0440     LDA INITDOS&255 ;PONER DOSINI
061A 850C           0450     STA DOSINI
061C A900           0460     LDA #INITDOS&255
061E 850D           0470     STA DOSINI+1
0620 A500           0480     LDA NUEVAM&255 ;SET MEMLO
0622 8DE702         0490     STA MEMLO
0625 A930           0500     LDA #NUEVAM/256
0627 8DE802         0510     STA MEMLO+1
062A 60             0520     RTS
062B                0530     *= $02E2
02E2 0E06           0540     .WORD DIREJE ; DISPONER DIRECCION DE EJECUCION.
02E4                0550     .END
```

FIGURA E.10 MODIFICADOR DE MEMLO

SECCION 8.5 - EL MONITOR

El monitor del Sistema Operativo es un programa en ROM, que maneja tanto la secuencia de energización como la de SYSTEM RESET. Las secuencias de energización y SYSTEM RESET son similares en sus funciones y de hecho comparten en gran parte el mismo código. El Apéndice IV contiene un diagrama de flujo de las rutinas de energización y SYSTEM RESET.

La rutina de energización (también conocida como partida en frío, coldstart) se invoca, ya sea energizando el computador o saltando a COLDSV [$E477]. COLDSV es el vector de rutina del sistema hacia la rutina de energización. Los puntos importantes que deben recordarse con respecto a la energización son:

1. Se limpia la totalidad de la memoria excepto las ubicaciones comprendidas entre $0000 y $000F.

2. Se intenta tanto la autocarga (boot) de cassette como de disco. BOOT? [$0009] es una bandera que indica el éxito o fracaso de las autocargas. Bit 0=1 indica una autocarga de cassette exitosa, Bit 1=1 una autocarga de disco exitosa.

3. COLDST [$0244] es una bandera, que le indica al monitor que se encuentra ejecutando una energización. COLDST = 0 significa SYSTEM RESET, COLDST<>0 significa energización. Un uso interesante de COLDST consiste en ponerlo a un valor diferente de 0 durante la ejecución de un programa de aplicación. Esto convertirá un SYSTEM RESET en una energización. Esta técnica puede agregar un grado de seguridad, previniendo al usuario de obtener control del computador, mientras está corriendo alguna aplicación.

Presionando la tecla SYSTEM RESET se produce una reposición del sistema (conocida también como partida en caliente). Algunos de los hechos claves que deben recordarse acerca del SYSTEM RESET del diagrama de flujo del Apéndice IV son:

1. Los vectores RAM del Sistema Operativo se copian de ROM tanto durante el SYSTEM RESET como la energización. Si se desea modificar un vector, deben tomarse algunas provisiones para administrar SYSTEM RESET. Para ello vea en el Capítulo 9.

2. MEMLO, MEMTOP, APPMHI, RAMSIZ, y RAMTOP se reponen durante la reposición del sistema. Si se quiere alterar estos punteros de RAM de usuario para reservar algo de espacio para módulos assembler llamados desde BASIC, debe tomarse alguna provisión para administrar el SYSTEM RESET. La Figura 8.10 da un ejemplo de cómo hacerlo.

SECCION 8.6 - LA BASE DE DATOS DEL SISTEMA

La base de datos del sistema contiene muchas ubicaciones que almacenan información de interés para el usuario. Las ubicaciones pueden accederse directamente desde programas de lenguaje assembler o desde BASIC con PEEK y POKE.

La base de datos del sistema ocupa la RAM de las páginas 0 a 4 ($0000-$03FF). La base de datos contiene banderas del sistema, memorias de transferencia entrada/salida, parámetros de entrada/salida, etc. El usuario puede recurrir a alguna de estas ubicaciones para proveer posibilidades no ofrecidas ni por el Sistema Operativo ni por el ambiente del programa (p.ej. BASIC). La Figura 8.11 describe algunos de estos elementos de la base de datos.

CUADRO DE LA BASE DE DATOS

Nombre	Ubicación ,Largo	Inic. por	Valor	Función
MEMLO	$02E7,2	E,R	x	límite inf. memoria usuario
MEMTOP	$02E5,2	E,R	x	límite sup. memoria usuario
APPMHI	$000E	E,R	$00	lím. inf. pantalla mem. us.
RAMTOP	$006A,2	E,R	x	lím. sup. RAM admin. pantalla
RAMSIZ	$02E4,1	E,R	x	dir. lím. sup. RAM (MSB)
POKMSK	$0010,1	E,R	x	copia habilitacion IRQ del S.O.
BRKKEY	$0111,1	E,R	$FF	bandera tecla BREAK; $FF indica que no hubo toque
IRQEN	$D20E	E,R	x	reg. bit habilit. IRQ de ANTIC
PTIMOT	$001C	E,R	x	tiempo lím. SIO impresor
CIOCHR	$002F		x	almac. temp. de CIO para función PUT de un carácter.
ZIOCB	$0020,16		x	bloque I/O pág. cero de CIO
BOOT?	$0009,1	E,R	x	bandera de boot; bit 0 es cassette; bit 1 es disco
CKEY	$004A,1	E	x	bandera monitor boot cassette
CASSBT	$004B	E		bandera boot de cassette
KBCODE	$D209,1		x	registro código de teclado
CH	$02FC	E,R	$FF	valor tecla actual
CH1	$02F2,1		x	último valor teclado
SHLOK	$02BE,1	E,R	$40	enclavam. SHIFT
KEYDEL	$02F1,1		$03	reloj softw. rebote teclas
SSFLAG	$02FF,1		x	bandera start/stop del despl.
ATRACT	$004D,1			Bandera rotac. colores
SRTIMR	$022B,1		x	reloj cuenta rotac. colores
COLDST	$0244,1			bandera de inicialización
WARMST	$0008,1			bandera de inicialización
CHBAS	$02F4,1	E,R	$E0	puntero a base caracteres
CRITIC	$0042	E,R		bandera de región crítica I/O

E significa Energización
R significa Reposición Sistema (SYSTEM RESET)
x significa que el valor varía según el caso

FIGURA 8.11 LA BASE DE DATOS DEL SISTEMA

PUNTEROS DE MEMORIA

El Sistema Operativo emplea cinco ubicaciones para mantener la pista de las memorias de usuario y de despliegue, MEMLO, MEMTOP, APPMHI, RAMTOP, y RAMSIZ. Las relaciones entre ellos aparecen indicadas en la Figura 8.12, un mapa de memoria simplificado del sistema.

MEMLO es una ubicación de dos bytes, que emplea el Sistema Operativo para indicar dónde puede iniciarse un programa de aplicación. Usado con cuidado, MEMLO puede emplearse para crear áreas para módulos en lenguaje assembler, que pueden llamarse desde BASIC. BASIC usa el valor de MEMLO para determinar la posición de partida de un programa (vea el Capítulo BASIC para los aspectos de estructura de un programa BASIC). Si deseamos poner MEMLO a una dirección mayor, debemos hacerlo antes de que se ejecute el cartridge BASIC. MEMLO debe usarse cuidadosamente, porque se repone tanto en energización como SYSTEM RESET.

Si se inicializa el sistema desde una unidad de disco, puede usarse la facilidad AUTORUN.SYS, para poner MEMLO a un valor predefinido. Como DOS también repone MEMLO durante SYSTEM RESET, a través del vector DOSINI, debemos modificar DOSINI. DOSINI contiene la dirección del código de inicialización DOS, que se llama como parte de la inicialización de sistema del monitor. DOS siempre será inicializado como parte de SYSTEM RESET. El contenido de DOSINI debe desplazarse a una dirección de dos bytes de una instrucción JSR como parte de la energización. DOSINI se pone a continuación a la dirección de la instrucción JSR del código c inicialización y MEMLO se pone al valor predefinido. Al ocurrir un SYSTEM RESET, hay un llamado al nuevo código de inicialización y la primera instrucción, JSR ANTDOSINI inicializa el DOS. El nuevo código de inicialización pone MEMLO en el valor predefinido y hace un RTS a la antigua secuencia de inicialización. La Figura 8.10 es un ejemplo de como hacerlo.

La técnica anterior también puede usarse con MEMTOP, el puntero de memoria de usuario superior. Puede disponerse espacio de memoria para módulos assembler y datos bajando MEMTOP con respecto a los valores puestos por la energización y SYSTEM RESET. El usar MEMTOP en vez de MEMLO crea un problema. MEMTOP fluctúa tanto con la RAM del sistema como con el modo gráfico del despliegue. Esto hace difícil predecir su valor antes de examinar realmente la ubicación, a no ser que se hagan algunas suposiciones con respecto al sistema. Como resultado puede suceder que todos los módulos de assembler deban ser reubicables.

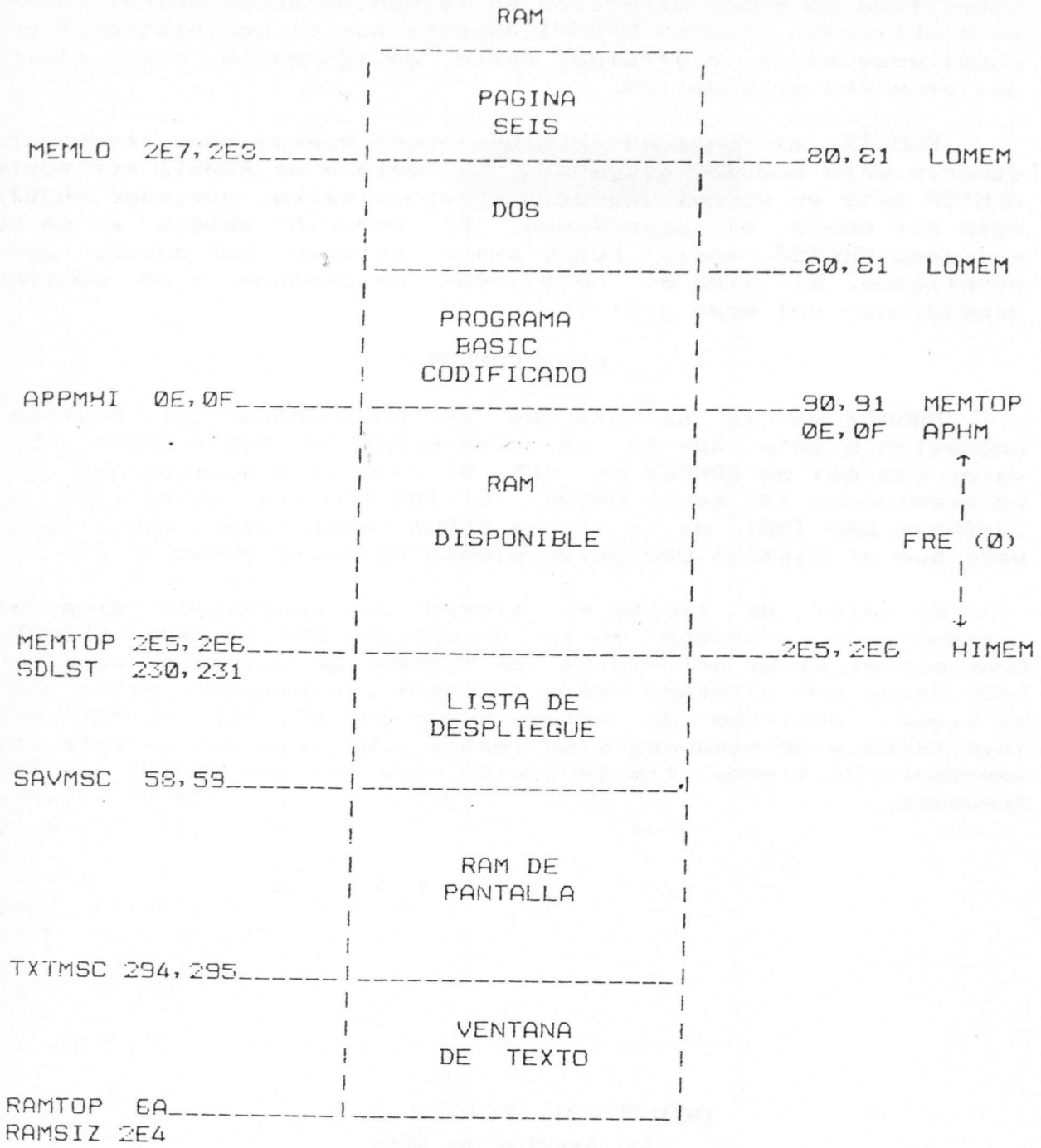
APPMHI es una ubicación que contiene una dirección que especifica la menor dirección en la que se puede ubicar la RAM de despliegue. Fijando APPMHI asegura que el administrador de despliegue no va a arruinar parte de los datos o del código del programa de usuarios.

RAMSIZ, al igual que MEMTOP, puede usarse para reservar espacio para módulos assembler. La ventaja de RAMSIZ por sobre MEMTOP está en que el espacio puesto a salvo moviendo RAMSIZ está por sobre el despliegue. El espacio puesto a salvo moviendo MEMTOP hacia abajo sigue estando por debajo del despliegue. El área de despliegue se expande y se contrae dependiendo del modo gráfico.

MISCELANEAS

BRKKEY es la bandera que se pone cuando el Sistema Operativo siente que se ha presionado la tecla BREAK. El valor nominal de BRKKEY es $FF. Si cambia, significa que se ha presionado la tecla BREAK. El IRQ (no la instrucción de software BRK IRQ) de la tecla BREAK debe estar habilitado para que el Sistema Operativo sienta la tecla BREAK.

El valor de límite de tiempo del computador para el impresor, se almacena en la ubicación RAM llamada PTIMOT. Contiene el valor del límite de tiempo, en segundos, para el SIO. Puede ser alterado para aumentar o disminuir el período de tiempo poniendo un valor nuevo en PTIMOT. PTIMOT se inicilaiza a 30 segundos y se repone cada vez que se abre el impresor. Un tiempo límite típico para el modelo 825 es de 5 segundos.



PUNTEROS DEL S.O. Y DEL BASIC (con DOS)

FIGURA 8.12

SECCION 8.7 - RELOJES

LOS RELOJES DEL SISTEMA

Se proveen dos tipos de relojes para el usuario. Los relojes del sistema corren a la frecuencia de cuadro de televisión. Para el sistema NTSC (usado en Chile, Norteamérica, Japón, etc.) la tasa es de 59,923334 hz. La televisión europea (PAL) corre a 50 Hz. El compás de los relojes POKEY se da por frecuencias definibles por el usuario.

Existen 8 relojes de sistema:

Nombre	Ubicación	Vector o bandera
RTCLOK	[$0012,3]	ninguna
CDTMV1	[$0218,2]	CDTMA1 [$0226,2]
CDTMV2	[$021A,2]	CDTMA2 [$0228,2]
CDTMV3	[$021C,2]	CDTMF3 [$022A,1]
CDTMV4	[$021E,2]	CDTMF4 [$022C,1]
CDTMV5	[$0220,2]	CDTMF5 [$022E,1]

Todos estos relojes de sistema se avanzan como parte del proceso de borrado vertical (VBLANK). Si el proceso de VBLANK se inhibe o intercepta, los relojes no marcharán.

El reloj de tiempo real (RTCLOK) y el reloj de sistema 1 (CDTMV1) se avanzan durante el VBLANK inmediato, etapa 1. RTCLOK cuenta a partir de 0 y es un valor de tres bytes. Cuando RTCLOK alcanza su valor máximo (16.777,216) se repone a 0. RTCLOK puede usarse como reloj de referencia, como lo muestra el ejemplo 8.13.

Debido a que los relojes del sistema se avanzan como parte del proceso VBLANK, debe tenerse especial cuidado para ponerlos en forma correcta. Para ello se usa una rutina de sistema llamada SETVBV [$E45C]. El llamado a SETVBV es:

```
Los registros:X,Y contienen los valores del reloj.
A contiene el número del reloj:
     A=1-5 - - RELOJES 1-5
```

Ejemplo:

```
        LDA #1          ; Poner reloj de sistema 1
        LDY #0
        LDX #02         ; el valor es de $200 VBLANKs
        JSR SETVBV      ; llamar rutina de servicio para
                          poner reloj
```

```
1 POKE 752,1:GOTO 3
3 ? CHR$(125):REM BORRAR PANTALLA
4 ? "HORA ";:INPUT HORA:? "MINUTO ";:INPUT MINUTO:
  ? "SEGUNDO ";:INPUT SEGUNDO
5 CMNDO=1:GOSUB 45
6 ? CHR$(125);HORA;" : ";MINUTO;" : ";SEGUNDO:? " ":? " "
7 CMNDO=2:GOSUB 45
9 POSITION 2,0:? HORA;" : ";MINUTO;" : ";SEGUNDO;"    ":GOTO 7
10 REM ESTA ES UNA DEMOSTRACION DE UN RELOJ DE TIEMPO REAL
20 REM ESTA RUTINA ACEPTA HORA,MINUTO Y SEGUNDO INICIALES
30 REM PONE EN CERO EL RELOJ DE TIEMPO REAL
40 REM EL VALOR DEL RELOJ DE TIEMPO REAL SE SUMA
   AL TIEMPO INICIAL PARA DAR EL TIEMPO ACTUAL
45 ALTO=1536:MEDIO=1537:BAJO=1538
50 REM
60 REM ***** PUNTO DE ENTRADA *****
70 ON CMNDO GOTO 100,200
95 REM
96 REM ** INICIALIZACION DEL RELOJ **
97 REM
100 POKE 20,0:POKE 19,0:POKE 18,0
105 DIM RELOJ$(50)
106 RELOJ$=" ":GOSUB 300
110 HORAIN=HORA:MINUTOIN=MINUTO:SEGUNDOIN=SEGUNDO
197 REM
198 REM ****** LEER RELOJ ******
199 REM
200 REM
201 A=USR(ADR(RELOJ$))
210 TIEMPO=((((PEEK(ALTO)*256)+PEEK(MEDIO))*256)
    +PEEK(BAJO))/59.923334
220 HORA=INT(TIEMPO/3600):TIEMPO=TIEMPO-(HORA*3600)
230 MINUTO=INT(TIEMPO/60):SEGUNDO=INT(TIEMPO-(MINUTO*60))
235 SEGUNDO=SEGUNDO+SEGUNDOIN:IF SEGUNDO>59 THEN
    SEGUNDO=SEGUNDO-60:MINUTO=MINUTO+1
236 MINUTO=MINUTO+MINUTOIN:IF MINUTO>59 THEN
    MINUTO=MINUTO-60:HORA=HORA+1
237 HORA=HORA+HORAIN
240 HORA=HORA-INT(HORA/24)*24
250 RETURN
300 FOR I=1 TO 38:READ Z:RELOJ$(I,I)=CHR$(Z):NEXT I:RETURN
310 DATA 104,165,18,141,0,6,165,19,141,1,6,165
320 DATA 20,141,2,6,165,18,205,0,6,208,234
330 DATA 165,19,205,1,6,208,227,165,20,205,2,6,208,220,96
```

FIGURA 8.13 - - UN RELOJ DE TIEMPO REAL

```
5 ? CHR$(125):REM   RUTINA BASIC PARA RITMO DEL METRONOMO
7 REM                    ORIGINAL DE M.EKBERG PARA CARLA
8 POKE 752,1
10 X=10:FOR I=1 TO 2 STEP 0
20 TOP=10:FOR J=1 TO TOP:NEXT J:REM BUCLE DE RETARDO
50 IF STICK(0)=13 THEN X=X+1:REM     CON BASTON ADELANTE
                                     SUBE RITMO
51 IF STICK(0)=14 THEN X=X-1:REM     CON BASTON ATRAS
                                     BAJA RITMO
52 IF X<1 THEN X=1:REM               NUNCA BAJAR A CERO
53 IF X>255 THEN X=255:REM           NUNCA PASAR DE 255
54 REM                               IMPRIMIR COMPASES POR
                                     MINUTO
55 POSITION 2,0
56 ? "";INT(3600/X);" COMPASES/MINUTO   "
60 POKE 0,X:REM                      EN LA UBICACION $0000 SE
                                     ENCUENTRA EL COMPAS
70 NEXT I:REM                        PARA LA SIGUIENTE RUTINA
ASSEMBLER.


40        *=$600
50 ;RUTINA METRONOMO...USA $0000 PARA COMPAS
60 ;
70 AUDF1 =    $D200    REGISTRO FREC. AUDIO
80 AUDC1 =    $D201    REGISTRO CONT. AUDIO
90 FREC  =    $08      VALOR DE AUDF1
0100 VOLUMEN = $AF     VALOR DE AUDC1
0110 CALLA = $A0       BAJAR VOLUMEN
0120 SETVBV = $E45C    PONER RUTINA VALOR RELOJ
0130 XITVBV = $E462
0140 CDTMV2 = $021A    RELOJ 2
0150 CDTMA2 = $0228    VECTOR RELOJ 2
0160 RELOJZ = $00      VALOR PAG. CERO RELOJ BORR. VERT.
0170 ;
0180 START LDA #10
0190      STA RELOJZ
0200 INIC
0210 ;PONER VECTOR RELOJ
0220 ;
0230      LDA #CUENTA&255
0240      STA CDTMA2
0250      LDA #CUENTA/256
0260      STA CDTMA2+1
0270 ;
0280 ;PONER RELOJ SEGUN VECTOR
0290 ;
0300      LDY RELOJZ   PONER RELOJ 2 PARA CONTAR
0310      JSR PONER
0320      RTS
```

```
0340 ; VECTORES CUENTA METRONOMO AQUI!
0350 ;
0360 CUENTA
0370 ;
0380 ; PONER CANAL DE AUDIO PARA CLIC METRONOMO
0390 ;
0400      LDA #VOLUMEN
0410      STA AUDC1
0420      LDA #FREC
0430      STA AUDF1
0433      LDY $FF      RETARDO
0437 RETARDO
0440      DEY
0450      BNE RETARDO
0460      JMP INIC
0470 ;
0480 ;
0490 ;          SUBRUTINA PARA PONER RELOJ
0500 ;
0510 PONER
0520      LDX #0       TIEMPOS MENORES QUE 256 INTS. VERTS.
0530      LDA #2       PONER RELOJ 2
0540      JSR SETVBV   RUTINA SISTEMA PARA PONER RELOJ
0550      RTS
0560      *=   $02E2
0570      .WORD START
0580      .END
```

FIGURA 8.14 - - METRONOMO

Los relojes de sistema 1-5 son contadores de dos bytes. Pueden ponerse a un valor, usando la rutina SETVBV. El Sistema Operativo los decrementa a continuación durante VBLANK. El reloj 1 se decrementa durante el VBLANK inmediato, etapa 1. Los relojes 2-5 se decrementan durante el VBLANK inmediato, etapa 2. El Sistema Operativo toma diferentes acciones según el reloj que haya decrementado a 0.

Los relojes de sistema 1 y 2 tienen vectores asociados. Cuando cualquiera de estos dos relojes llega a 0 el Sistema Operativo simula un JSR a través del vector del reloj dado. La Figura 8.9B da los vectores para ambos relojes.

Los relojes de sistema 3-5 tienen banderas que normalmente están puestas (SET diferente de 0). Cuando cualquiera de estos tres relojes baja su cuenta a 0, el Sistema Operativo va a limpiar (poner en 0) la bandera de ese reloj. El usuario puede por lo tanto verificar la bandera y tomar la acción apropiada.

Los relojes de 1-5 son relojes de software para propósitos generales que pueden usarse para una serie de aplicaciones. Por ejemplo el SIO emplea el reloj 1 para acompasar las operaciones del bus serial. Si el reloj llega a 0 con su cuenta antes de haberse completado la operación, se retorna un error de tiempo excedido. El reloj 1 se pone a diferentes valores dependiendo del dispositivo accedido. Esto asegura de que por un lado mientras un dispositivo siempre tendrá tiempo suficiente para contestar a un requerimiento de entrada/salida, por otro lado el computador no se quedará esperando indefinidamente a que conteste un dispositivo no existente. El reloj 3 también lo usa el Sistema Operativo. El administrador de cassette emplea el reloj 3 para disponer la duración de tiempo para los encabezamientos de lectura y escritura de cinta. El ejemplo 8.14 recurre al reloj 2 para entregar un sonido, que se puede usar como un metrómono. La tasa del metrónomo puede fijarse en diversos valores recurriendo a un bastón.

CAPITULO 9
EL SISTEMA OPERATIVO DE DISCO
(D.O.S. 2.0)

INTRODUCCION

El Sistema Operativo de Disco (DOS) es una extensión de Sistema Operativo, que permite el acceso al almacenamiento masivo de las unidades de disco como archivo. Se puede acceder archivos de discos igual como cualquier otro tipo de archivo. Discutamos las partes del DOS primero, a continuación discutiremos como usar el DOS.

El Sistema Operativo de Disco (DOS) tiene tres partes, el administrador residente de disco, el manejador de archivos (File manager system, FMS) y el paquete de utilitarios de disco (DUP). El administrador residente de disco es la única parte del DOS que se encuentra en la ROM del Sistema Operativo. El FMS y el DUP se encuentran en el diskette y se cargan (autocargan) en el computador al energizarlo.

EL ADMINISTRADOR RESIDENTE DE DISCO

El administrador residente de disco es la parte más simple del DOS. El administrador de disco no se conforma a la secuencia normal de llamado del CIO como los demás administradores de dispositivos. La relación del administrador de disco con el subsistema de entrada/salida se muestra en la Figura 8.1 de la Sección 8.2 de este libro.

De la Figura 8.1 vemos que el DCB es el camino para comunicar con el administrador de disco. La secuencia de llamado para el administrador de disco es:

```
                    ; quien llama ha dispuesto el DCB
JSR DSKINV          ; vector de rutina del sistema al
                      administrador residente de disco
BPL OKAY            ; ramificacion si hubo exito, reg.Y=1
                    ; caso contrario, reg.Y=Status del error
                      (DCBSTA también contiene el error)
```

El administrador soporta cinco funciones:

FORMAT (FORMATEAR)	Editar un comando de formateo al controlador de disco
READ SECTOR (LEER SECTOR)	Leer un sector específico
WRITE SECTOR (WRITE SECTOR)	Escribir un sector específico
WRITE/VERIFY SECTOR (ESCRIBIR/VERIFICAR SECTOR)	Escribir un sector y verificar si fue escrito
STATUS (ESTADO)	Preguntar al controlador de disco por su estado (status).

El comando FORMATEO limpia todas las pistas del diskette y las direcciones de sectores se escriben nuevamente sobre las pistas. Con este comando, no se graba ninguna estructura de archivo en el diskette.

Los tres comandos de entrada/salida de sector pueden usarse para leer y escribir sectores en el diskette. Se puede usar para implementar una estructura de archivos propia. La Sección 10 del manual del Sistema Operativo contiene un ejemplo de uso del administrador de disco para escribir un archivo de autocarga.

La función STATUS se emplea para determinar el estado de la o las unidades de disco. Puede usarse el comando STATUS para varios propósitos. Como el tiempo límite del comando STATUS es menor que él de los demás comandos, se puede usar para verificar si determinada unidad de disco se encuentra conectada. Si el administrador de disco retorna un error de tiempo excedido del dispositivo, ya se sabe que esa unidad de disco no está conectada.

DUP

El DUP (Paquete de utilitarios de disco) es un conjunto de utilitarios para diskette, que se ve normalmente como el menú del DOS 2.0. El DUP ejecuta los comandos llamando el FMS a través del CIO. Los comandos son:

A. DIRECTORY	DIRECTORIO
B. RUN CARTRIDGE	EJECUTAR CARTRIDGE
C. COPY FILES	COPIAR ARCHIVOS
D. DELETE FILES	ELIMINAR ARCHIVOS
E. RENAME FILES	REBAUTIZAR ARCHIVOS
F. LOCK FILES	PROTEGER ARCHIVOS
G. UNLOCK FILES	DESPROTEGER ARCHIVOS
H. WRITE DOS FILES	ESCRIBIR ARCHIVO DOS
I. FORMAT DISK	FORMATEAR DISCO
J. DUPLICATE DISK	DUPLICAR DISCO
K. SAVE BINARY FILES	GRABAR ARCHIVO BINARIO
L. LOAD BINARY FILE	CARGAR ARCHIVO BINARIO
M. RUN AT ADDRESS	EJECUTAR EN DIRECCION
N. WRIRE MEM.SAV FILE	CREAR ARCHIVO MEM.SAV
O. DUPLICATE FILE	DUPLICAR ARCHIVO

FMS

El FMS (Sistema de manejo de archivo) es un administrador de dispositivo no residente que hace uso de la interfaz de administrador de dispositivo-CIO normal. El FMS no está presente en la ROM del Sistema Operativo. Se autocarga durante la energización del computador, si se encuentra presente un diskette que contenga el DOS.

El FMS, al igual que los demás administradores de dispositivos, obtiene sus datos de control entrada/salida del CIO. A continuación el FMS recurre al administrador de disco residente para hacer su entrada/salida al diskette. Se llama al FMS disponiendo un IOCB y llamando el CIO. El FMS soporta algunas funciones especiales del CIO, no disponibles para los otros administradores:

FORMATEO (FORMAT)	El FMS llama al administrador residente de disco para formatear el diskette. Después de un formateo exitoso, el FMS escribe algunos de los datos de estructura de archivo sobre el diskette.
ANOTAR (NOTE)	El FMS retorna el valor actual del puntero del archivo
APUNTAR (POINT)	El FMS pone el puntero del archivo a un valor especificado.

ENTRADA/SALIDA DE DISCO

Se puede acceder a todos los llamados de entrada/salida de archivos normales a través del CIO. Esto significa que en BASIC se usan los comandos de entrada/salida, como OPEN, CLOSE, GET, PUT. En lenguaje assembler debe disponerse el bloque IOCB y hacer un llamado al CIO. Usemos BASIC para dar una introducción simple al DOS.

Para realizar cualquier proceso de entrada/salida al disco, en primer lugar debe abrirse (OPEN) un archivo. La sintaxis de BASIC para el OPEN es:

```
OPEN #IOCB,ICAX1,0,"D:MIPROG.BAS"
```

Con # IOCB se escoje uno de los 7 IOCB disponibles en BASIC (BASIC mismo usa el IOCB #0). ICAX1 es el código para OPEN. Los bits para los diferentes códigos son:

BIT	7	6	5	4	3	2	1	0
	x	x	x	x	E	L	D	A

En que: A es Agregar
D es Directorio
L es lectura
E es escritura
x es sin uso

Los diferentes valores de ICAX1 se discuten en la Sección 5 del manual del Sistema Operativo. Algunos de los puntos claves que deben recordarse acerca de los diversos modos OPEN son:

ICAX1=6 Se usa para abrir el directorio del diskette. Los registros leìdos serán los ingresos al directorio del diskette.

ICAX1=4 Modo de lectura (READ).

ICAX1=8 Modo de escritura (WRITE). Cualquier archivo abierto (OPEN) en este modo se elimina. Los primeros bytes escritos estarán en el punto inicial del archivo.

ICAX1=9 Modo de escritura de agregados (WRITE APPEND). El archivo permanece intacto. Los bytes escritos a este archivo se ubican al final del archivo.

ICAX1=12 Modo de puesta al dia (UPDATE). Este modo permite tanto lecturas (READ) como escrituras (WRITE) al mismo archivo. Los bytes leìdos o escritos se ubican en el lugar del primer byte del archivo.

ICAX1=13 No soportado.

Ahora que sabemos como abrir (OPEN) un archivo, veamos como transferir datos entre nuestro programa y el disco. Hay dos modos de entrada/salida que pueden usarse: registro o carácter.

La entrada/salida de carácter significa simplemente que los datos del archivo forman una lista de bytes. El DOS interpreta esta lìnea de bytes como datos (sin caracteres de control embebidos). Este es un ejemplo de datos de carácter (todos los valores son hexadecimales): 00 23 4F 55 FF 34 21.

Entrada/salida de registro significa que los datos del archivo se forman como un juego de registros. Un registro es un grupo de bytes limitados por un EOL ($9B). El siguiente es un ejemplo de dos registros:

```
00 23 4F 55 FF 34 9B 21 34 44 29 F4 9B
|    registro 1    |   registro 2    |
```

Entrada/salida de registro y carácter pueden hacerse en cualquier orden. De hecho, los datos creados como registros pueden leerse como caracteres. Datos de archivo creados como caracteres pueden leerse como registros. La única diferencia entre entrada/salida de caracteres y de registros está en que los registros terminan en $9B. $9B se considera como un dato corriente al usar entrada/salida de carácter.

El BASIC soporta bastante bien la entrada/salida por registro. Los comandos PRINT e INPUT pueden usarse para escribir y leer registros de archivo. BASIC no soporta entrada/salida de caracteres en forma tan buena como podrìa. Los comandos GET y PUT permiten leer y escribir bytes individuales cada vez.

El Sistema Operativo posee la capacidad de leer y escribir bloques de caracteres. BASIC no aprovecha esta habilidad. Además del largo del bloque, el Sistema Operativo permite la especificación de la dirección del bloque. Para usar el modo de bloques de caracteres del Sistema Operativo desde BASIC, puede escribirse un módulo en lenguaje Assembler que se llamar desde BASIC por medio de la funcion USR. La Figura 8.6 de la Sección 8.2 de este libro, da un ejemplo de subrutina para lograr entrada/salida de bloques de caracteres.

ACCESO ALEATORIO

Uno de los usos más importantes del diskette corresponde al acceso aleatorio de archivos almacenados en orden arbitrario. El uso de los comandos de entrada/salida, conjuntamente con los comandos especiales NOTE y POINT permite la creación y el manejo de archivos de acceso aleatorio.

NOTE y POINT leen y ponen al dìa el puntero del archivo respectivamente. DOS mantiene un puntero de archivo para cada archivo, que en el momento esté abierto (OPEN), que indica al DOS la ubicación actual del archivo. El puntero de archivo tiene dos parámetros, el número de sector y la cuenta de bytes. El número de sector (un valor comprendido entre 1 y 719) le indica al DOS a cual sector del diskette está apuntando el puntero de archivo. La cuenta de bytes es una cuenta dentro del sector en el que nos encontramos (es decir, el primer byte del sector tiene la cuenta byte 0, el segundo byte corresponde a 1, etc.). La Figura 9.1 muestra la relación del puntero de archivo con el archivo. Todos los valores son hexadecimales. Los valores del puntero de archivo para los bytes de este archivo se dan debajo de los bytes del archivo.

				E O D				E O						E O		
	A	B	C	L	D	E	F	L	G	H	I	J	K	L	A	B
Archivo	41	42	43	9B	44	45	46	9B	47	48	49	4A	4B	9B...41		42
Puntero																
Sector No.	50	50	50	50	50	50	50	50	50	50	50	50	50	50...50		51
Byte No.	0	1	2	3	4	5	6	7	8	9	A	B	C	D...7C		0

El archivo precedente se creó en BASIC con:

```
10 OPEN #1,8,0,"D:ARCHIVO.DAT
20 ?#1;"ABC"
30 ?#1;"DEF"
40 ?#1;"GHIJ"
 :
 :                    :REM Llenar todo el sector
 :
100 ?#1;"AB"          :REM Aquí está el registro que cruza el
final del sector No. 50
150 CLOSE #1
```

Figura 9.1 VALORES DE NOTE Y POINT

El número de sector es 50 porque el DOS comenzó arbitrariamente con este archivo en el sector 50. El número de sector cambia a 51 porque el archivo es de mayor longitud que un sector. DOS enlazó el archivo al siguiente sector disponible, el 51. El registro "AB" transpasa el final del primer sector.

La cuenta de bytes del puntero de archivo comienza en 0 y se incrementa hasta el final del sector, $7D (125 Dec.) El DOS reserva los últimos 3 bytes de cada sector para datos generales del archivo. Para los archivos bajo DOS 2.0, la cuenta máxima de bytes es hasta 124 (0 a 124=125 bytes en total). Cuando un archivo llega al final de un sector, la cuenta de bytes se repone a 0.

A esta altura, Ud. debería tener ya una buena idea de como se archivan los registros en un diskette y como se recuperan. La Figura 9.2 es una subrutina para grabar registros, mantener su pista y recuperarlos. El Apéndice VIII contiene un método de acceso aleatorio completo escrito en BASIC.

```
1000 REM esta rutina escribe, lee y modifica registros de largo
     invariable en un archivo de acceso aleatorio.
1010 REM los comandos son
1020 REM CMDO=1 escribir registro N
1030 REM CMDO=2 leer registro N
1040 REM CMDO=3 modificar registro N
1050 REM
1060 REM REGISTRO$ es el registro de entrada/salida
1070 REM N es el numero del registro
1080 REM INDICE es un agrupamiento bidimensional de
     DIM INDICE(1,CANTREG).
1090 REM INDICE contiene los valores NOTE de todos
     los registros.
1100 REM SEC=0 y BYTE=1
1110 REM apertura y cierre del archivo se realizan en
     el programa ppal.o en otra rutina
1200 ON CMDO GOTO 2000,3000,4000
2000 REM
2010 REM escribir registro N, al final del archivo existente
2020 REM
2030 NOTE #1,X,Y
2040 INDICE(SEC,N)=X:INDICE(BYTE,N)=Y
2050 ? #1;REGISTRO$:RETURN
3000 REM
3010 REM leer registro N
3020 REM
3030 X=INDICE(SEC,N):Y=INDICE(BYTE,N)
3040 POINT #1,X,Y
3050 INPUT #1;REGISTRO$
3060 RETURN
4000 REM
4010 REM modificar registro N
4020 REM
4040 X=INDICE(SEC,N):Y=INDICE(BYTE,N)
4050 POINT #1,X,Y
4060 ? #1;REGISTRO$
4070 RETURN
```

FIGURA 9.2 EJEMPLOS DE NOTE Y POINT

CAPITULO 10
SONIDO

1. Las capacidades circuitales de los ATARI.

Los microcomputadores ATARI tienen grandes capacidades de sonido. Poseen 4 canales de sonido, controlables independientemente. Cada canal tiene un registro de frecuencia que determina la nota, un registro de control que ajusta el volumen y el contenido de ruido. Hay varias opciones que permiten insertar filtros pasa-altos, escoger las bases de reloj, disponer modos alternativos de operación y modificar los poli-contadores.

Para los efectos de este capìtulo, deben aclararse algunos términos:

1 Hz (Hertz)	corresponde a un pulso por segundo
1 kHhz (kilo-Hertz)	corresponde a 1.000 pulsos por segundo
1 MHz (mega-Hertz)	corresponde a 1.000.000 de pulsos por segundo

De aquì en adelante, un pulso se define como un repentino incremento de voltaje, seguido de una caìda igualmente repentina, que se producen al interior de un circuito electrónico y que si se envìa al parlante del televisor por ejemplo, se convierte en un golpe audible (varios pulsos en sucesión se convertirán en un zumbido, etc.).

Una "onda", tal como se usa aquì, se refiere a una variación de voltaje con respecto al tiempo y también describe un tipo de sonido; es decir, las ondas creadas por los computadores ATARI son ondas rectangulares (como las de la Figura 10.2); los instrumentos de bronce producen ondas triangulares y un cantante produce ondas sinusoidales (ilustradas en la Figura 10.15). Si se envìan al parlante del televisor, estos voltajes se convertirán en ondas sonoras.

Un ciclo de onda rectangular es lo mismo que el pulso descrito más arriba, y una cadena continua de pulsos viene siendo una onda rectangular.

Un registro de desplazamiento es como un byte de memoria (en el sentido de que retiene un dato binario), el cual cuando asì se le indique desplazará todos sus bits hacia la derecha en una posición, es decir, el bit 5 contendrá ahora lo que habìa en el bit 4, en el bit 4 se encontrará lo que haya habido en el bit 3, etc. De este modo, el bit de más de la derecha cae afuera y el de más a la izquierda toma el valor que habìa en su lìnea de entrada:

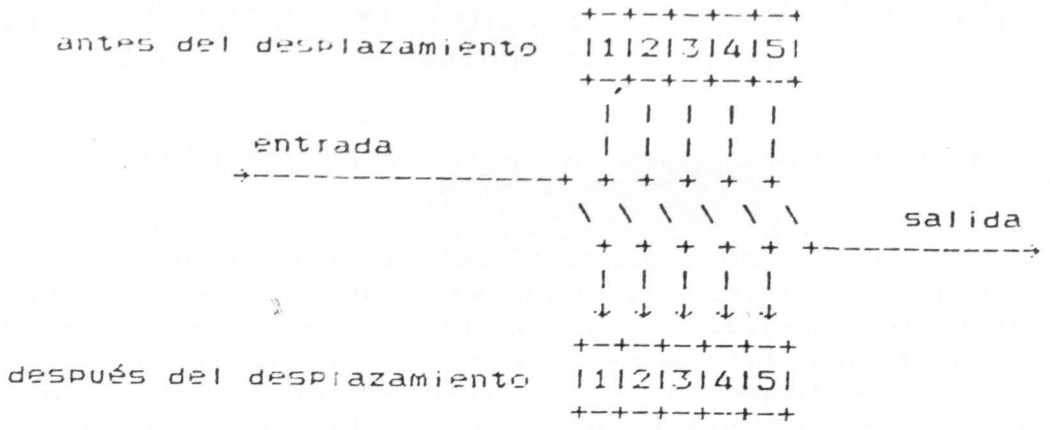


Figura 10.1
Diagrama del flujo de bits en un registro de desplazamiento

AUDF1-4 debe leerse como: "cualquiera de los registros de frecuencia de audio entre 1 y 4". Sus direcciones son: $D200, $D202, $D204, $D206 (53760, 53762, 53764, 53766).

AUDC1-4 debe enterderse como: "cualquiera de los registros de control de audio del 1 al 4". Sus direcciones son: $D201, $D203, $D205, $D207 (53761, 53763, 53765, 53767).

Las palabras "frecuencia", "nota", "tono", y "altura" se usan indiferentemente. Para los propósitos de este capìtulo su significado es el mismo.

"Ruido" y "distorsión" también se usan indiferentemente, aunque sus significados no sean exactamente iguales. En realidad, "ruido" es una descripción más acertada de la función que realiza el ATARI.

La interrupción de 60 Hz, a la que se hace referencia en la parte II de este capìtulo, se llama también la interrupción del borrado vertical.

Todos los ejemplos están en BASIC, a no ser que se indique de otra forma. Digite los ejemplos exactamente como aparezcan, es decir, si no hay número de lìnea, no lo ponga, y si varias sentencias se encuentran en una misma lìnea, digìtelos igual.

Encontrará algunos problemas al hacer POKE con sonidos en BASIC o lenguaje de máquina. Debe inicializarse el circuito POKEY para que funcione bien. Para inicializarlo desde BASIC, ejecute una sentencia de sonido nulo, es decir, SOUND 0,0,0,0. En lenguaje de máquina, almacene un 0 en AUDCTL ($D208=53768), y un 3 en SKCTL ($D20F=53775).

AUDF1-4

Cada canal de audio tiene su correspondiente registro de frecuencia que controla la nota tocada por el computador.

El registro de frecuencia contiene el número "N" usado en un circuito divisor por N. Esta división no lo es en el sentido matemático, sino algo bastante más simple: por cada N pulsos que ingresan, egresa uno solo. Por ejemplo, más abajo tenemos el diagrama de una función que divide por cuatro:

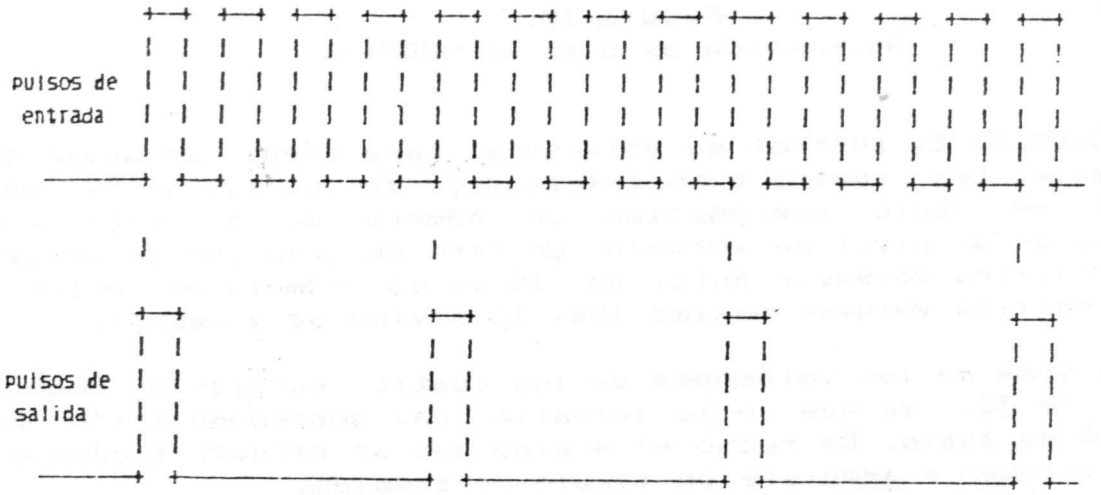


Figura 10.2
Diagrama de división por 4

A medida que N crece, los pulsos de salida se harán menos frecuentes, siendo la nota más grave.

Para los propósitos de esta discusión, la frecuencia es una medida para el número de pulsos que ocurran en un tiempo dado, es decir, que una nota tenga una frecuencia de 100 Hz significa que en un segundo ocurren exactamente 100 pulsos. Mientras más frecuentes (de ahí el término "frecuencia") los pulsos de una nota, más alta será la nota. Por ejemplo, una cantante soprano canta a frecuencias altas (unos 5 kHz) y una vaca muge a frecuencias bajas (unos 100 Hz).

AUDC1-4

Cada canal posee también su correspondiente registro de control. Estos registros permiten ajustar el volúmen y el contenido de distorsión de cada canal. La asignación de bits para AUDC1-4 es como sigue:

AUDC1-4

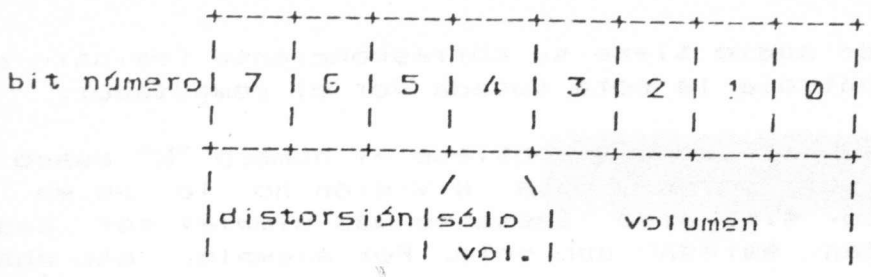


Figura 10.3
Asignación de bits de AUDC1-4

VOLUMEN: El control de volumen de cada canal de audio es muy simple. Los cuatro bits inferiores de los registros de control de audio representan un número de 4 bits que corresponde al nivel de volumen. Un cero en cada uno de estos bits significa volumen nulo, un 15 en el número de estos 4 bits significa volumen máximo (Hay 16 nivles de volumen).

La suma de los volúmenes de los cuatro canales no deberìa exceder de 32, ya que ello forzarìa una sobremodulación del circuito de audio. De hecho el sonido que se produce tiende a perder volúmen y adquiere un timbre de zumbido.

DISTORSION: Cada canal tiene también 3 bits de control de distorsión en su registro de control de audio. La distorsión se usa para crear efectos sonoros especiales, cada vez que un tono puro sea poco deseable.

El aprovechamiento de la distorsión que logra ATARI es único en la industria. Su versatilidad y controlabilidad lo prestan muy bien para la sìntesis de una variedad casi infinita de sonidos, desde ronquidos, tarreos y chillidos hasta clics, murmullos y ruidos de fondo, establecedores de ambientes con silbidos de fantasmas o provenientes desde el espacio exterior.

La distorsión, tal como se define en ATARI, no es equivalente a la interpretación normal. Por ejemplo, "distorsión de intermodulación" y "distorsión armónica" son criterios cualitativos especificados para sistemas estereofónicos de alta fidelidad. Estos tipos de distorsión se refieren a la degeneración de las formas de onda, cuando la forma de una onda se altera un poco debido a imperfecciones en la circuiterìa electrónica. La distorsión de ATARI no altera las ondas (siempre son ondas cuadradas), sino más bien elimina pulsos seleccionados de la forma de onda total. Esta técnica no se describe adecuadamente por la palabra "distorsión". Un término más apropiado y descriptivo para la distorsión en el sentido de ATARI serìa "ruido".

Sin embargo, antes de poder comprender plenamente las técnicas de generación de ruidos ATARI, es necesario entender los poli-contadores.

Los poli-contadores se emplean en los ATARI como fuente de pulsos aleatorios en la generación de ruidos. Los poli-contadores ATARI consisten en lo principal de un registro de desplazamiento (vea la descripción de un registro de desplazamiento al comienzo de este capìtulo), que desplaza muy rápidamente (1,79 MHz). La salida del registro de desplazamiento entrega los puntos aleatorios y su entrada se determina procesando los valores de bits seleccionados del registro de desplazamiento.

Por ejemplo, en el diagrama que sigue, el valor antiguo del bit 5 se empujará fuera del registro para convertirse en el siguiente pulso aleatorio, y el bit 1 será una función de los bits 3 y 5:

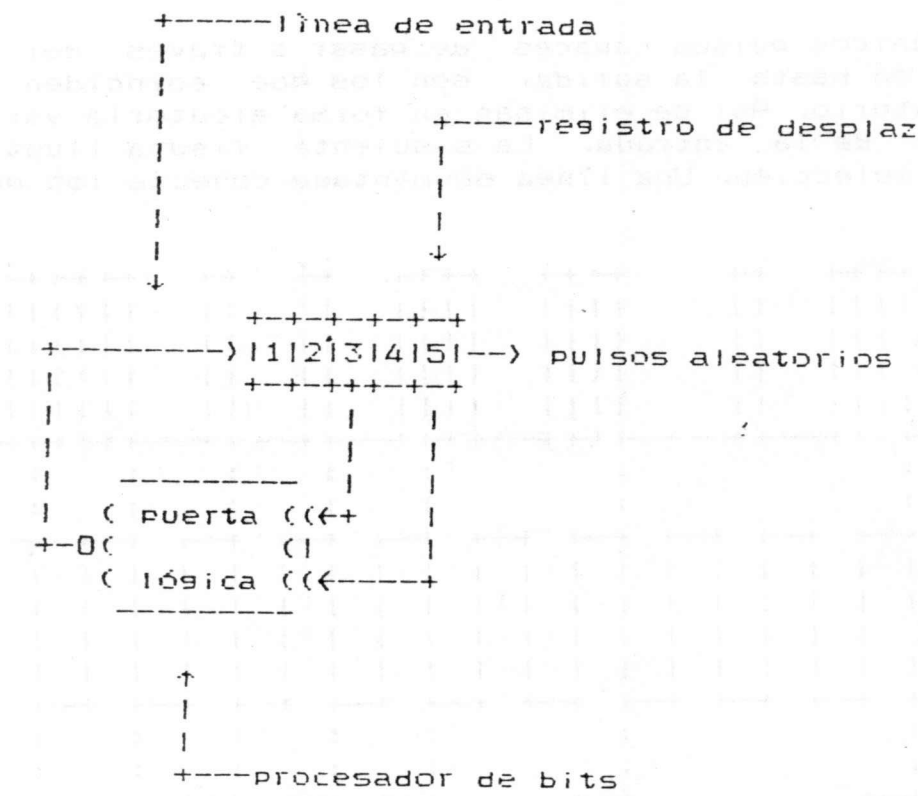


Figura 10.4
Poli-contador de 5 bits

El procesador de bits recibe valores de ciertos bits del registro de desplazamiento (bits 3 y 5, indicados más arriba) y los procesa en una forma que es irrelevante para esta

discusión. Entrega un valor que a su vez se convierte en el bit número 1 del registro de desplazamiento del poli-contador.

Estos poli-contadores no son en realidad verdaderamente al azar, ya que repiten su secuencia de bits después de un cierto lapso de tiempo. Como puede sospecharse, su tasa de repetición dependerá del largo (en número de bits) del registro de desplazamiento que se use en el poli-contador; es decir, uno de mayor largo requerirá muchos desplazamientos antes de empezar a repetirse, mientras los más cortos comenzarán a repetir con mayor frecuencia.

Con esta base en materia de poli-contadores, estamos ahora equipados para entender como el ATARI genera la distorsión.

En los ATARI la distorsión se obtiene, usando pulsos aleatorios provenientes de estos poli-contadores, en un circuito de selección. Este circuito, de hecho, es un comparador digital, pero la palabra "circuito de selección" es más descriptiva, de modo que la usaremos en este capìtulo.

Los únicos pulsos capaces de pasar a través del circuito de selección hasta la salida, son los que coinciden con un pulso aleatorio. Asì se eliminan en forma aleatoria varios de los pulsos de la entrada. La siguiente figura ilustra el método de selecciòn. Una lìnea segmentada conecta los pulsos que coinciden:

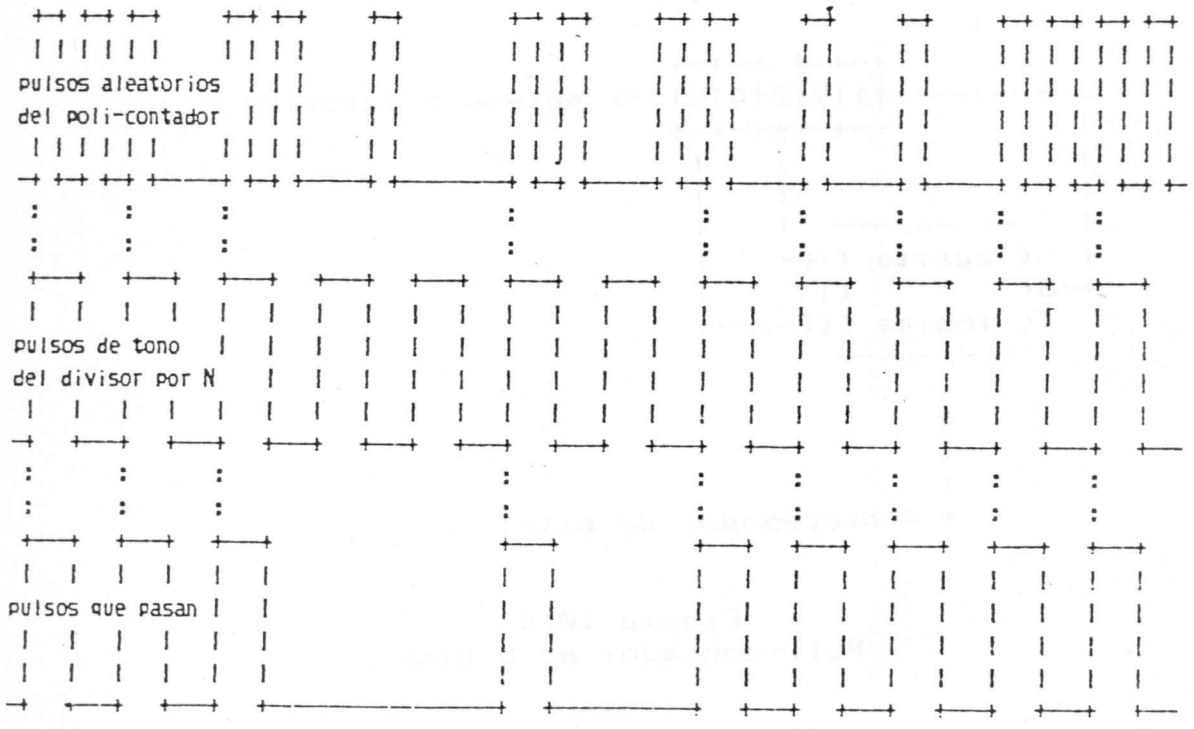


Figura 10.5
La función de selección usada para introducir distorsión

El efecto final es este: algunos de los pulsos del circuito divisor de frecuencia (discutido en la parte I, en la sección de título AUDF1-4) se eliminan. Obviamente, si se eliminan varios pulsos, la nota sonará de otra forma. Es así como se introduce la distorsión en un canal de audio.

Como los poli-contadores repiten su secuencia de bits, también se repetirá una cierta configuración de pulsos. Y como el circuito de selección usa esta configuración para eliminar pulsos, la nota distorsionada tendrá también esta misma configuración. Es este el método que permite al programador crear zumbidos de zánganos, motores y otros que tengan configuraciones repetitivas.

Los ATARI están equipados de tres poli-contadores de diferentes largos, que dan diferentes niveles de azar. Los poli-contadores más pequeños (de 4 y 5 bits de largo respectivamente) repiten con suficiente frecuencia como para crear sonidos de zumbidos que suben y caen rápidamente, mientras el poli-contador más largo (de 17 bits de largo) toma tanto tiempo para comenzar a repetirse que no se puede observar periodicidad en esta distorsión. Este poli-contador de 17 bits puede usarse para explosiones, vapor y cualquier sonido con crujidos y estallidos al azar. Incluso es suficientemente irregular para usarse en la generación de ruido blanco (un término de audio que se refiere a un sonido sibilante). Para obtener un ejemplo de la generación de ruido blanco, ensaya lo siguiente:

```
SOUND 0,100,9,8
POKE 53768,64
```

Una opción de poli-cortador de 9 bits ofrece un compromiso razonable entre los poli-contadores cortos y el largo (vea la parte I, discusión de AUDCTL).

Cada canal de audio ofrece 5 combinaciones diferentes de estos tres poli contadores:

AUDC1-4

```
+-+-+-+-+-+-+-+-+
|7|6|5|4|3|2|1|0|
+-+-+-+-+-+-+-+-+
 ↑ ↑ ↑
 ↓ ↓ ↓
+-----+
|0 0 0| reloj/frec., selecc. con poli-5 y poli-17, div. por 2
|0 X 0| reloj/frec., selecc. con poli-5, div.por 2
|0 1 0| reloj/frec., selecc. con poli-5 y poli-4, div. por 2
|1 0 0| reloj/frec., selecc. con poli-17, div. por 2
|1 X 1| reloj/frec., div. por 2, (sin poli-contadores)
|1 1 0| reloj/frec., selecc. con poli-4, div. por 2
+-----+
```

NOTAS: "Reloj" quiere decir frecuencia base - vea la discusión de AUDCTL en la parte I

Una 'X' significa: "no importa si este bit está puesto o no. La estructura de AUDC1- 4, tal como se muestra en la Figura 10.7, aclara porque es así.

Figura 10.6
Las posibles combinaciones de poli-contadores

Estos bits superiores de AUDC1-4 controlan tres conmutadores en el circuito de audio indicado más abajo. El diagrama también le ayudará a entender porqué la tabla de la Figura 10.6 está estructurada en esta forma:

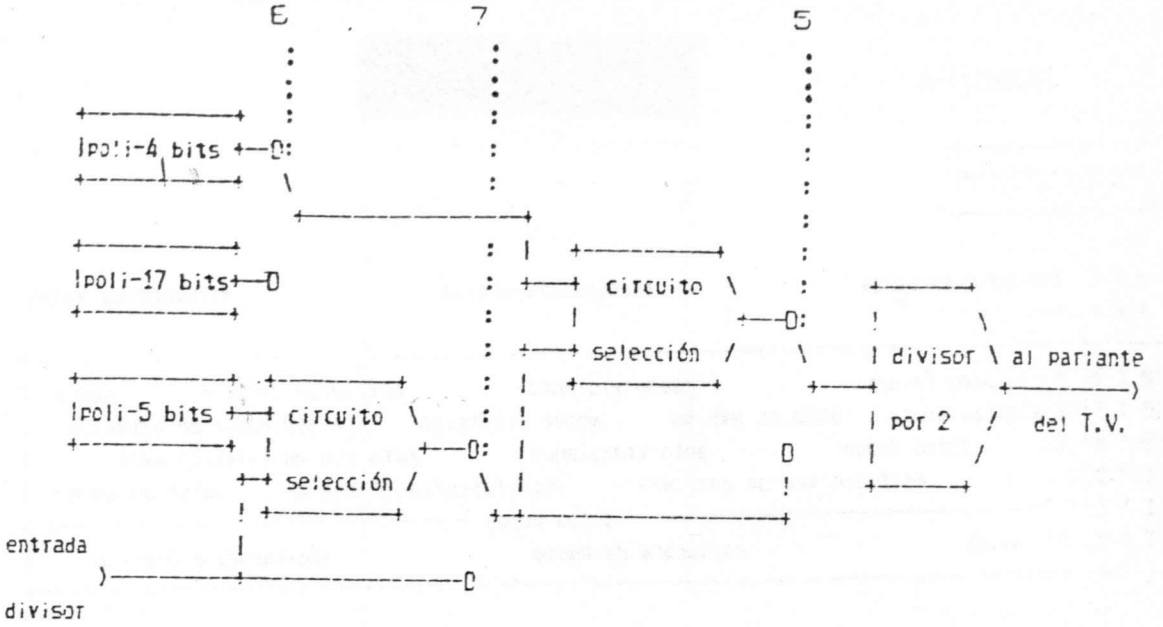


Figura 10.7
Diagrama en bloques de AUDC1-4

Cada combinación de los poli-contadores otorga un sonido único. Cuando busque un sonido en particular, ensaye cada combinación de poli-contadores a diferentes frecuencias, ya que la distorsión dispuesta puede dar sonidos completamente diferentes dependiendo de la frecuencia. Más abajo hay una tabla de suposiciones, solamente para comenzar:

AUDC1-4

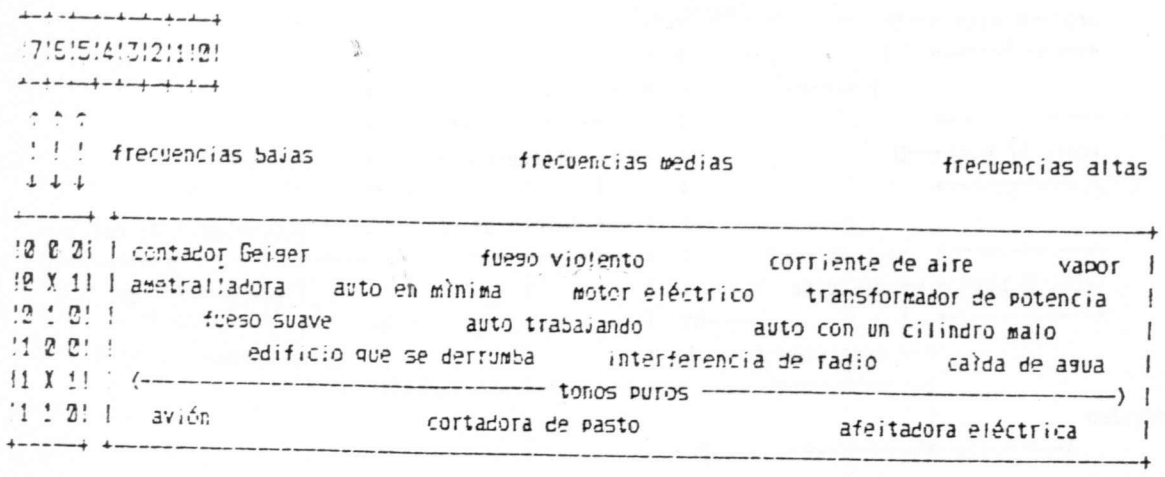


Figura 10.8
Sonidos producidos por combinaciones de distorsión a diferentes frecuencias

Sumario de distorsión: se usa un registro de desplazamiento como componente principal de un poli-contador, el que a su vez se emplea para generar pulsos aleatorios. Los pulsos aleatorios se usan para eliminar pulsos seleccionados de las notas, introduciendo así una distorsión en el canal. Los bits superiores (bits 7,6,5) de AUDC1-4 conmutan tres interruptores que seleccionan los poli-contadores que se usarán para la distorsión en ese canal.

VOLUMEN SOLAMENTE: El bit # 4 de AUDC1-4 especifica el modo de "volúmen solamente". Cuando se pone este bit, los bits de volúmen (AUDC1-4 bits 0-3) se envían al televisor como volúmen, es decir, sin que se asocie una frecuencia a estos bits.

Para comprender bien el uso de este modo de operación, debe entenderse como funciona un parlante y que le sucede a un parlante (de televisor) cuando recibe un pulso.

Los parlantes tienen un cono que se mueve hacia afuera y hacia adentro. La posición del cono en cualquier instante es directamente proporcional al voltaje que recibe en ese momento desde el computador: es decir, si el voltaje es 0, el

parlante estará en su posición de reposo, si el voltaje es 5, el parlante estará en una posición extendida. Cada vez que la posición del cono cambia, mueve el aire, lo cual el oído lo detecta como sonido.

A partir de nuestra definición de pulso, sabemos que éste consiste de una subida de voltaje seguida de una caída de voltaje. Si tuviéramos que mandar un pulso al parlante, éste adelantaría su cono con el aumento de voltaje y lo volvería hacia atrás con la caída de voltaje, resultando así una onda de aire que puede ser detectada por nuestro oído como un golpe. La siguiente sentencia producirá un golpe de este tipo en el parlante del televisor, al enviar un solo pulso:

```
POKE 53761,31:POKE 53761,16
```

Un tren de pulsos (o una onda) pondría al parlante en constante movimiento y se escucharía un zumbido continuo. Es así como el computador genera sonido a través del parlante del televisor.

Es esencial comprender que el volúmen que se envía no cae a 0, sino que se mantiene hasta que el programa lo altere. Se espera que el programa module el volúmen con suficiente frecuencia como para crear un ruido. Ensaye ahora lo siguiente, escuchando cuidadosamente después de cada sentencia:

```
POKE 53761,31
POKE 53761,31
```

La primera vez se escuchó un golpe, lo cual era de esperar: es decir, el parlante se movió hacia afuera y movió el aire. Pero la segunda vez no hubo sonido. Ello es porque el parlante ya se encuentra en la posición exterior — un nuevo comando de extensión no le hace nada al parlante, ni mueve el aire, de manera que no se escucha nada.
Ahora escuche esto:

```
POKE 53761,16
POKE 53761,16
```

Tal como antes, se escucha un golpe la primera vez que el parlante se mueve hacia atrás a su posición de reposo, y nada se escucha la segunda vez, ya que el parlante se encuentra ya en la posición de reposo.

Así, el bit de "volumen solamente" lo otorga al programa un control completo sobre la posición del parlante en cualquier momento. A pesar de que los ejemplos anteriores son

solamente ejemplos binarios (todo o nada), de ninguna manera la modulación del parlante está limitada a algo de este tipo. De hecho se permite forzar el parlante a cualquiera de 16 posiciones diferentes.

Por ejemplo, una onda triangular (similar a la forma de onda producida por los instrumentos de bronce) puede generarse enviando un volumen de 8 seguido por 8,9,10,11,10,9,8,7,6,5,6,7, y vuelta a 8 y repitiendo esta secuencia una y otra vez rápidamente (BASIC es demasiado lento). En este ejemplo se usan 7 de las 15 posibles posiciones del parlante (el programa ejemplo, dado en la sección de generación de sonido con código de máquina, de hecho produciría una onda triangular, si su tabla de duraciones contuviera solamente valores 1).

Por ello, cambiando rápidamente el volumen, puede crearse virtualmente cualquier forma de onda. Es posible, por ejemplo, hacer decir "hola" al computador en voz clara.

Si Ud. nunca ha sabido de la teoría de las ondas de audio o electrónicas, el propósito y uso del bit de" volúmen solamente" pueden parecerle bastante confusos. Probablemente lo mejor sería que se consiguiera ahora un libro sobre esta materia.

Hay una discusión completa acerca de este bit en la parte II.

AUDCTL

Fuera de los bytes de control de canal independientes (AUDC1-4), hay un byte opcional (AUDCTL), que afecta a todos los canales. Cada bit de AUDCTL tiene una función específica:

AUDCTL ($D208 = 53768) si este bit está puesto...

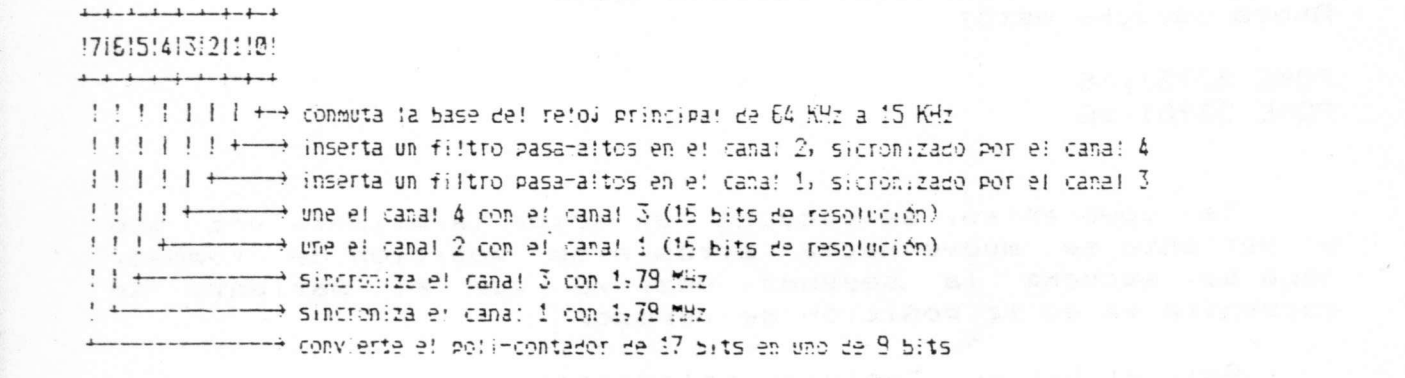


Figura 10.9
Asignación de bits de AUDCTL

LA SINCRONIZACION: Antes de continuar con las explicaciones de las opciones de AUDCTL, debe entenderse muy bien lo que es la sincronización. En general, las millones de minúsculas operaciones internas, que ocurren cada segundo en un computador, se sincronizan con el tren de pulsos de un reloj. Un reloj late continuamente y cada pulso le indica a la circuitería, que debe realizar otro paso en sus operaciones.

En las sección sobre frecuencia se explicó, que la salida de un divisor de frecuencia es de solamente uno por cada N pulsos. Pero, de dónde provienen los pulsos de la entrada? (Del reloj!

En el computador se utilizan varios relojes y AUDCTL permite cambiar el reloj en uso para la entrada para dividir por N y así obtener un reloj más rapido o más lento. Cuando cambie este reloj de entrada también cambiará la salida del divisor de freceuncia proporcionalmente.

Por ejemplo, imaginemos que el reloj esté latiendo a una tasa de 15 Khz y el registro de frecuencia está puesto para dividir por 5. La tasa de los pulsos de salida del circuito divisor será de 3 KHz. Pero si cambiáramos el reloj, o la frecuencia de entrada, a 40 KHz sin cambiar el registro de frecuencia, que pasaría? El divisor por N siempre estaría entregando uno de cada 5 pulsos, pero está recibiendo entradas a una tasa mayor y así cada quinto pulso llega en forma más rápida. El resultado será una frecuencia de salida (del divisor por N) de 8 KHz.

La fórmula para la frecuencia de salida (del divisor por N) es muy simple:

$$\text{frecuencia de salida} = \frac{\text{reloj}}{N}$$

Esto demoestra que la frecuencia de salida es directamente proporcional al reloj de entrada.

Así, si alguien acelerara el reloj, todo lo que use ese reloj se ejecutará con menor retardo. En el caso de la circuitería de audio de los computadores ATARI, si aumenta la tasa del reloj, todos los pulsos de sonido que van al televisor estarían mas próximos uno al otro, haciendo crecer la frecuencia o el tono (vea en la parte I, la sección sobre frecuencia).

Lo inverso es igualmente cierto - si el reloj se frenara, todos los pulsos se separarían uno de otro,

reduciendo la freceuncia.

Opción de 15 KHz: Pruebe lo siguiente:

```
SOUND 0,128,10,8          tono medio
POKE 53.768,1             opción de 15 KHz de AUDCTL
```

Como se explicó en la sección de sincronización, un reloj más lento produce una cadena menos apretada de pulsos de salida o un sonido más grave. La ubicación 53768 es AUDCTL y un 1 pone el bit #0, él que conmuta el reloj de 64 KHz a uno de 15 KHz. En la demostración precedente, Ud. escuchó como la frecuencia caía aproximadamente 1/4 de la frecuencia original, ya que el reloj cayó a 1/4 de su tasa original.

Es importante anotar que si este bit está puesto, todos los canales de sonido que eran sinconizados por los 64 KHz, se cambiarán a una base de reloj de 15 KHz.

OPCIONES de 1,79 MHz Pruebe lo siguiente:

```
SOUND 2,255,10,8          hace funcionar el canal 1, tono
grave
POKE 53768,64             Poner bit #6 de AUDCTL
```

Como se explicó en la sección de sinconizacion, cambiar la base de tiempo, cambia proporcionalmente la frecuencia que se escucha. En este caso, poniendo un 64 en AUDCTL hace que el canal #1 use la base de tiempo de 1,79 MHz en vez de la de 64 KHz. Como era de esperar de la sección previa, el POKE hizo que el tono se hiciera más agudo. El cambio produce una diferencia sustancial, ya que 1,79 MHz es casi 30 veces más rápido que 64 Khz.

Poniendo el bit # 5 de AUDCTL obliga al canal # 3 a usar 1,79 MHz como reloj en la misma forma:

```
SOUND 2,255,10,8          hace funcionar el canal 3, tono
                          grave
POKE 53768,32             Poner bit #5 de AUDCTL.
```

Estas opciones extienden el rango de frecuencia que puede generarse, hasta más allá de los límites del oído humano (podemos escuchar hasta cerca de 20 KHz).

OPCIONES DE 16 BIT: Los bits 3 y 4 de AUDCTL logran juntar 2 canales, creando un canal con un rango dinámico extendido. Trabajando independientemente, la frecuencia de cada canal puede variar en un rango de 0 a 255 (8 bits de capacidad de división por N). Juntando 2 canales se logra un rango de frecuencia de 0 a 65535 (una capacidad de 16 bits del divisor por N). En este modo es posible reducir la frecuencia de salida a simples "pops" separados por varios

segundos. El siguiente programa usa dos canales en el modo de 16 bits y dos paletas de control como entradas de frecuencia. Inserte un juego de paletas en la puerta # 1, digite y ejecute el siguiente programa:

```
10 SOUND 0,0,0,0                          Inicializar sonido
20 POKE 53768,80                          Sincronizar canal 1 con
                                          1,79 MHz, sincronizar
                                          canal 2 con canal 1
30 POKE 53761,160:POKE 53763,168          cortar canal 1, conectar
                                          canal 2 (tonos puros)
40 POKE 53760,PADDLE(0):POKE 53762,PADDLE(1)
50 GOTO 40                                se usan las paletas para
                                          poner los valores de
                                          frecuencia en sus
                                          registros
```

La paleta derecha ajusta el sonido en forma gruesa y la paleta izquierda lo ajusta en forma fina entre los incrementos gruesos.

Este programa, en primer lugar pone los bits 4 y 6 de AUDCTL, lo cual significa "sincronizar el canal 1 con 1,79 Mhz y juntar canal #2 con canal #1". Una vez que esto suceda, los registros de frecuencia de 8 bits de ambos canales, se supone que representan un solo número de 16 bits N, usado para dividir el reloj de entrada.

A continuación, el volumen del canal #1 se pone en 0. Esto, porque la salida del canal 1 no tiene ningún significado, salvo para el canal 2. Cuando estos dos canales se ponen en su modo de 16 bits, cambios internos hacen que ambos canales funcionen como un circuito divisor de 16 bits. Estos cambios inhiben la salida del primer canal.

El registro de frecuencia del canal # 1 se usa como el byte fino o bajo dentro de la generación de sonido, y el registro de frecuencia del canal #2 es el byte grueso o superior. Por ejemplo, haciendo un POKE de 1 en el registro de frecuencia del canal 1, hace que el par divida por 1. Haciendo un POKE de 1 al registro de frecuencia del canal 2, hace que el par divida por 256; haciendo POKE 1 en ambos registros hace que el par divida por 257.

El bit #3 de AUDCTL puede usarse para unir los canales 4 y 3 en exactamente la misma forma.

La opción de 16 bits es útil en aplicaciones en las cuales se requiere un control más fino de la frecuencia. Estos incrementos más finos se consiguen usando el reloj de entrada de 1,79 MHz y usando una de las opciones de 16 bits como en el programa precedente en la línea 20.

Para entender el porqué los incrementos son más finos, debe entenderse lo que sucede con la frecuencia de salida a medida que el valor en el registro de frecuencia cambia. Primero un ejemplo concreto: si el registro de frecuencia esta dividiendo por uno y la frecuencia de entrada es de 1 KHz, la salida sera de 1 KHz. Ahora, incrementemos el registro de frecuencia, de modo que divida por 2, con lo cual obtenemos una salida de 500 Hz, lo cual es una disminución de 500 Hz. Incrementemos una vez más el registro de frecuencia, de modo que divida por 3. La salida será ahora de 333 Hz, una caída de 267 Hz. El segundo incremento tiene un efecto menor que el primero (el primer incremento produjo un cambio de 500 Hz mientras que el segundo incremento sólo afectó en 267 Hz). Si siguiéramos incrementando, cada paso tendría menor efecto que el precedente.

La idea básica es como sigue: mientras mayor sea el número N en los registros de división de frecuencia, menor será la diferencia en frecuencia de salida a medida que N cambie.

Por ello, mientras sincronizar el canal de 16 bits con 1,79 MHz y enseguida dividiendo por varios miles, dará la misma frecuencia de salida que un canal de 8 bits sincronizado con 64 Khz y dividiendo por varios cientos, el canal de 16 bits ofrecerá una resolución más fina, ya que cada incremento (o decremento) produce un efecto menor.

Para tener un poco de entretención en 16 bits, probemos las siguientes sentencias y experimentemos con varias combinaciones de valores en los 4 últimos POKE:

```
SOUND 0,0,0,0
POKE 53768,24
POKE 53761,168
POKE 53763,168
POKE 53765,168
POKE 53767,168
POKE 53760,240:REM intente POKE con otros valores en las
                  siguientes cuatro ubicaciones
POKE 53764,252
POKE 53752,28
POKE 53756,49
```

FILTROS PASA-ALTOS: Los bits 1 y 2 de AUDCTL controlan los filtros pasa-altos de los canales 2 y 1 respectivamente. Un filtro pasa-altos es un filtro en el sentido que permite que solamente cosas determinadas pasen a través de él. Como lo implica su nombre, las cosas que pasan son las frecuencias más altas.

En el caso de estos filtros pasa-altos, las frecuencias altas se definen como cualquier cosa más alta que el reloj. El reloj en este caso es la salida de cualquier otro canal.

Por ejemplo, si el canal #3 está generando mugidos de vaca y se halla puesto el bit #2 de AUDCTL, solamente un ruido o sonido con frecuencia más alta que la del mugido se podrá escuchar en el canal 1 (cualquier cosa más grave que el mugido será filtrado):

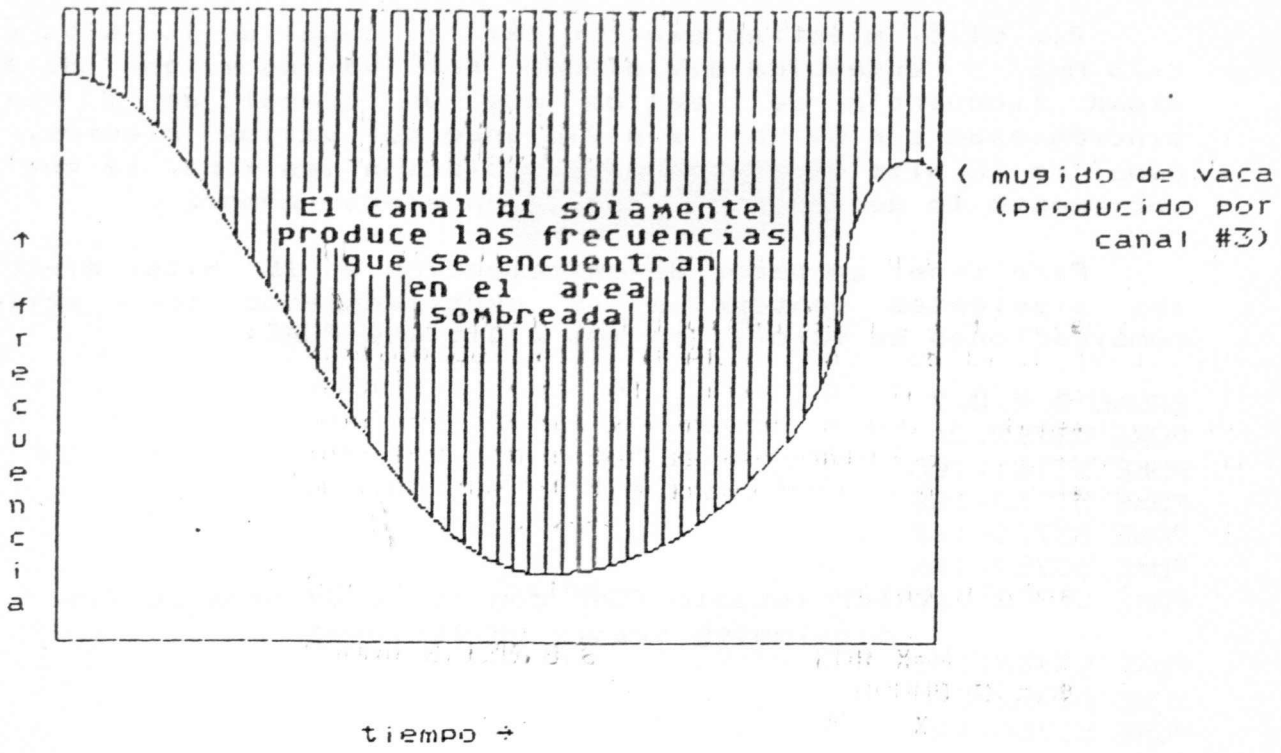


Figura 10.10
Diagrama del efecto de un filtro pasa-alto que se inserta en el canal 1 y está sincronizado por el canal 3

Se puede programar el filtro en tiempo real, ya que se usa el reloj de otro canal, que puede cambiarse en cualquier momento. Esto abre un gran campo de posibilidades al programador.

Los filtros se usan principalmente para crear efectos especiales. Pruebe lo siguiente:

```
SOUND 0,0,0,0
POKE 53768,4
POKE 53761,168:POKE 53765,168
POKE 53760,254:POKE 53764,127
```

CONVERSION DEL POLI-CONTADOR DE 9 BIT: El bit # 7 de AUDCTL convierte el poli-contador de 17 bits en uno de 9 bits. En la discusión de los poli-contadores se explicó que mientras más corto sea éste, más frecuentemente se repite la distorsión o tanto más discernible será el tipo de distorsión. Por ello, es razonable suponer que, cambiando el poli-contador de 17 bits en uno de 9 bits, se hará maá definible la caracterìstica del ruido.

Ensaye la siguiente demostración de la opción del poli-contador de 9 bits, escuchando cuidadosamente cuando se ejecute el POKE:

```
SOUND 0,80,8,8                    usar poli-17
POKE 53768,128                    cambiar a poli-9
```

II. Técnicas de programaciòon para la generación de sonidos.

Existen dos formas básicas de aprovechar los sistemas de sonido de los ATARI. La estática y la dinámico. Estático significa que el programa dispone algunos generadores de sonido, espera y enseguida los detiene. Dinámico quiere decir que los generadores de sonido se ponen al dìì continuamente, a medida que el programa se ejecuta. Por ejemplo:

Sonido estático

```
SOUND 0,120,8,8
```

sonido dinámico

```
FOR X=0 TO 255
  SOUND 0,X,8,8
NEXT X
```

Sonido estático:

El sonido estatico es bastante limitado. En lo principal, solamente pueden realizarse sonidos como pitos, clics y zumbidos. Sin embargo, hay algunas excepciones a esta regls. Dos ejemplo son los programas dados como efectos especiales en la parte I, sección de filtros pasa-altos y sección de sonido de 16 bits. Otro ejemplo más simple consiste en usar dos canales de sonido en la siguiente forma:

```
SOUND 0,255,10,8
SOUND 1,254,10,8
```

El extraño efecto resultante se debe a las puntas y a los valles muy próximos. Examine el siguiente diagrama. Muestra dos canales independientes, que llevan ondas sinusoidales de frecuencias levemente diferentes y tambien su suma. La curva suma muestra la extraña interferencia que se crea cuando se suman estos dos canales.

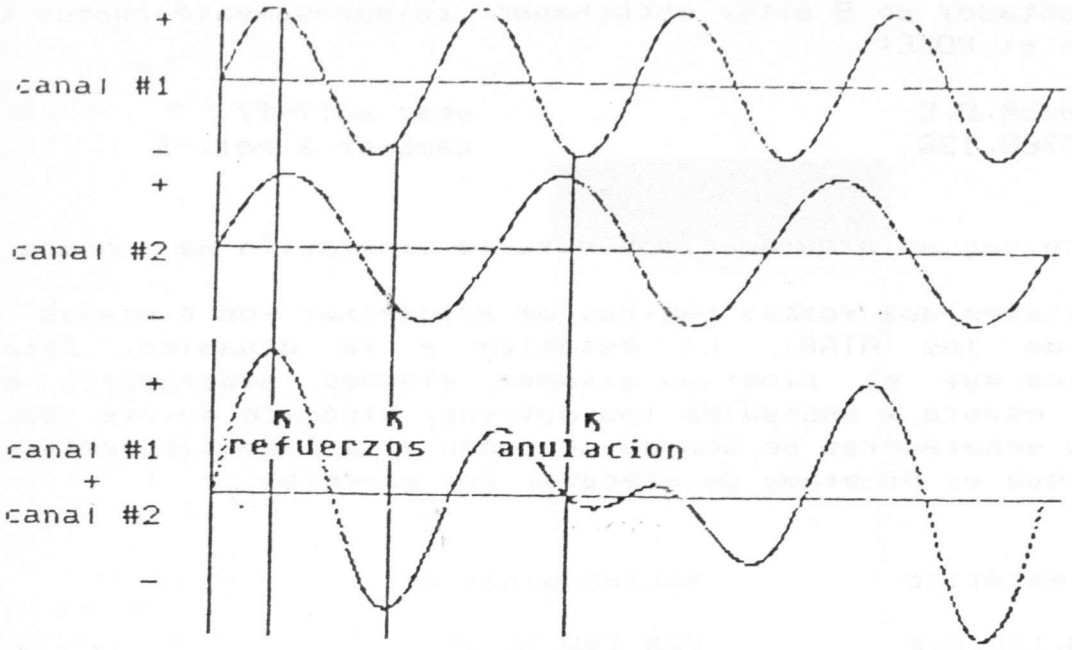


Figura 10.11
Dos ondas sinusoidales de diferentes frecuencias y su suma

Antes de enviar los dos canales al televisor, se mezclan uno con otro. El proceso de mezcla es similar a la mezcla realizada por el oído humano, una simple adición.

La Figura 10.11 muestra que en algunos instantes las ondas se refuerzan, mientras que en otros puntos ellas se oponen. Agregando los volúmenes de ambas ondas cuyas puntas coinciden, dará una onda del doble de fuerza o volumen. Igualmente, agregando los volúmenes de dos ondas mientras una esté en su máximo y la otra en su mínimo, resultará en una cancelación de ambas.

En el gráfico de la curva que da la suma, podemos ver este efecto. Hacia los extremos del gráfico, el volumen crece, ya que las puntas y los valles de ambos canales se acercan mucho unos a otros, casi duplicando el sonido. Mientras, al centro del diagrama las ondas se oponen una a otra y la onda resultante es plana. Podría ser interesante escribir un programa para trazar las curvas de interacción de una, dos, tres y cuatro canales tales como los de la Figura 10.11. Posiblemente descubrirá sonidos bastante singulares.

Mientras menor sea la diferencia en frecuencia entre los dos canales, más prolongada será la curva. Para entender esto, trace Ud. algunos gráficos similares a los de la Figura 10.11 y estudie la interacción. Como ejemplo, ensaye las siguientes sentencias:

```
SOUND 0,255,10,8
```

```
SOUND 1,254,10,8
SOUND 1,253,10,8
SOUND 1,252,10,8
```

Hay muchos otros ejemplos similares, usando el mismo principio:

```
SOUND 0,254,10,8
SOUND 1,127,10,8
```

Sonido dinámico

En general, cualquier sonido que no sea simplemente un pito, un clic o un zumbido, debe generarse a través del sonido dinámico. El programador ATARI tiene a su disposición tres niveles de sonido dinámico: sonido en BASIC, interrupción de 60 hz. y sonido en código de máquina.

SONIDO BASIC: El BASIC es un tanto limitado en sus capacidades de sonido. Como habrá notado, cualquier sentencia SOUND anula una disposición modificada de AUDCTL.

Este problema potencial puede evitarse, haciendo POKE para los sonidos, en vez de emplear la sentencia SOUND. El programa demostrativo dado en la parte I, bajo opciones de 16 bits de AUDCTL, constituye un buen ejemplo de esta técnica.

Adicionalmente, BASIC tiene sus límites a raíz de su baja velocidad. Si el programa no está totalmente dedicado a la generación de sonido, generalmente no hay suficiente tiempo de procesador para crear más que un sonido estático o un sonido dinámico entrecortado. La alternativa es detener temporalmente todo otro tipo de procesamiento, mientras se genera el sonido.

Puede presentarse otro problema al usar el computador para tocar música en más de un canal. Mientras se empleen los cuatro canales, la separación de tiempo entre la primera sentencia de sonido y la cuarta, puede ser suficientemente importante como para hacer notable el retardo inevitable entre la melodía y la armonía (los acordes).

He aquí una solución a este dilema:

```
10 SOUND 0,0,0,0:DIM SIMUL$(15)
20 RESTORE 9999:X=1
25 READ Q:IF Q<>-1 THEN SIMUL$(X)=CHR$(Q):X=X+1:GOTO 25
27 RESTORE 100
30 READ F1,C1,F2,C2,F3,C3,F4,C4
40 IF F1=-1 THEN END
50 X=USR(ADR(SIMUL$),F1,C1,F2,C2,F3,C3,F4,C4)
55 FOR X=0 TO 150:NEXT X
60 GOTO 30
100 DATA 182,168,0,0,0,0,0,0
110 DATA 162,168,182,166,0,0,0,0
120 DATA 144,168,162,166,35,166,0,0
130 DATA 128,158,144,266,40,166,35,166
140 DATA 121,158,128,166,45,166,40,166
150 DATA 108,168,121,166,47,156,45,166
160 DATA 96,168,108,166,53,166,47,166
170 DATA 91,168,96,166,60,166,53,166
999 DATA -1,0,0,0,0,0,0,0
9000 REM
9010 REM
9020 REM estos DATA contienen el programa en lenguaje de
9030 REM máquina y se leen a SIMUL$
9999 DATA
104,133,203,162,0,104,104,157,0,210,232,228,203,208,
     246,96,-1
```

En este programa, SIMUL$ es un pequeño programita en lenguaje de máquina, que hace "pokes" de los cuatro canales de sonido en forma virtualmente simultánea. Todo programa que use SIMUL$ puede poner en fase en forma acertada los cuatro canales.

Cualquier programa puede llamar a SIMUL$ simplemente poniendo los valores de los registros de sonido al interior de la función USR tal como se muestra arriba en la línea 50. Los parámetros deben ordenarse en la forma indicada, con los valores de los registros de control siguiendo los valores de los registros de frecuencia y repitiendo este orden una a cuatro veces, una vez para cada canal de sonido que quiera disponerse.

En consideración a la velocidad como a la conveniencia, SIMUL$ permite especificar sonido también para menos de cuatro canales; es decir 1 y 2 y 3, o 1 y 2, o simplemente 1. Para ello no ponga los parámetros sin uso al interior de la función USR. A continuación se muestra un llamado a SIMUL$ con los dos primeros canales solamente:

```
X=USR(ADR(SIMUL$),F1,C1,F2,C2)
```

SIMUL$ ofrece otra clara ventaja al programador BASIC.

Como ya se dijo, el registro AUDCTL se repone con la ejecución de cualquier sentencia SOUND en BASIC. Sin embargo, usando SIMUL4 no se ejecuta ninguna sentencia SOUND, de modo que la disposición de AUDCTL se mantiene.

Existe otra forma muy poco práctica de generar sonido en BASIC. Consiste en usar el bit de "volumen solamente" de cualquiera de los cuatro registros de control de audio. Digite y ejecute lo siguiente:

```
SOUND Ø,Ø,Ø,Ø
1Ø POKE 53761,16:POKE 53761,31:GOTO 1Ø
```

Este programa dispone el bit de volumen solamente del canal # 1 y modula el sonido de Ø a 15 tan rápido como BASIC puede hacerlo! En términos prácticos: este programa requiere el 1ØØ% del tiempo de procesamiento disponible y solamente produce un suave zumbido.

La mejor forma de obtener sonidos complejos en BASIC, sin estar ligado totalmente al procesador, es usando la técnica de interrupción de 50 Hz, descrita en la sección siguiente.

INTERRUPCION DE 50 Hz: Esta técnica sea probablemente la más versátil y práctica de todos los métodos que tiene a su disposición el programador ATARI.

Exactamente cada 50 avo de segundo, la circuitería del computador, en forma automática, genera lo que se llama una interrupción. Cuando esto sucede, el computador temporalmente abandona su programa principal (el programa que está ejecutando el sistema: por ejemplo BASIC, STAR RAIDERS (tm)). A continuación ejecuta una rutina de servicio de interrupción: una pequeña rutina, diseñada específicamente para servir estas interrupciones. Cuando termina la rutina de servicio de interrupción, ejecuta una instrucción especial en lenguaje de máquina que repone el computador en el programa interrumpido. Todo esto ocurre en forma tal (si esta bien hecho), que la ejecución del programa no se afecta, y de hecho no tiene idea de que haya sido detenida.

La rutina de servicio de interrupción, que normalmente se encuentra en los ATARI, simplemente mantiene los relojes, translada información de control y atiende otras cosas misceláneas que requieren una atención regular.

Pero antes de que la rutina de servicio de interrupción devuelva el control al programa principal, puede indicársele que ejecute alguna rutina de usuario: así, por ejemplo, la rutina de generación de sonido. Esto es ideal para la generación de sonido, ya que la sincronización está muy bien controlada, y especialmente porque puede ejecutarse cualquier otro programa sin que tenga que pagarse tributo al generador de sonido.

Aún más impresionante es su versatilidad. Un programa de interrupción de sonido se presta igualmente bien para un programa principal escrito en cualquier lenguaje - BASIC, Assembler, FORTH, PASCAL. De hecho, el generador de sonido requerirá muy pocas si es que algunas modificaciones para trabajar con otros programas e incluso con otro lenguaje. Una rutina comandada por una tabla ofrece el máximo de flexibilidad y simplicidad para un propósito de este tipo. Lo de comandado por tabla se refiere a un tipo de programa que accesa tablas de datos que se encuentren en memoria para su información. En el caso de un generador de sonido, las tablas de dato contendrían los valores de frecuencia y posiblemente los valores del registro de control de audio. La rutina simplemente leería la siguiente entrada de la tabla de datos y los pondría en los respectivos registros de audio. Mediante este método, las notas podrían cambiarse hasta 60 veces por segundo, lo cual es suficiente para la mayoría de las aplicaciones.

Este programa deberá estar escrito en lenguaje de máquina, ya que de hecho se convierte en una parte del sistema operativo.

Una vez que se haya escrito y ubicado en memoria un programa de este tipo (digamos, en la ubicación $6000) es necesario instalarlo como parte de la rutina de servicio de interrupción de 60 Hz. Esto se consigue con un método conocido como modificación de vector y se describe con mayor detalle en el apéndice I.

Las ubicaciones de memoria $224,$225 contienen la dirección de una pequeña rutina llamada XITVBL (eXIT Vertical BLank interrupt service routine). XITVBL está diseñada para ser ejecutada cada vez que el procesamiento de la interrupción de 60 Hz se haya completado, reponiendo al computador en su programa principal tal como se discutió anteriormente.

El proceso para instalar la rutina de sonido será como sigue:

1) Ponga su programa en memoria
2) Verifique que la última instrucción a ejecutar sea un JMP $E462. ($E462 es XITVBL, de modo que esto hará continuar el programa principal.
3) Cargue el registro X con el byte mayor de la dirección de su rutina (un 6 en este caso).
4) Cargue el registro Y con el byte inferior de la dirección de su rutina (un 0 en este caso.)
5) Cargue el acumulador con un 7.
6) Haga un JSR $E45C (para disponer las ubicaciones $224,$225).

Los pasos 3-6 anteriores se requieren para cambiar el valor de $224, $225 sin error. La rutina que se llama es SETVBV (SET Vertical Blank Vectors), la que simplemente pondrá la dirección de su rutina en las ubicaciones $224,$225 (vea el apéndice I).

Una vez instalado, el sistema trabajará tal como se indica a continuación, cada vez que ocurra alguna interrupción:

1) Se ejecuta la rutina de interrupción del computador.
2) Salta al programa cuya dirección se encuentra en $224,$225, que es ahora nuestra rutina.
3) Se ejecuta nuestra rutina.
4) Nuestra rutina ahora salta a XITVBL.
5) XITVBL repone el computador y hace que reinicie su operación normal.

Para efectos de comparación, lo siguiente muestra una secuencia de interrupción, sin que nuestra rutina forme parte de ella:

1) El computador ejecuta la rutina de interrupción.
2) Salta a la dirección especificada en $224,225, la cual es XITVBL.
3) XITVBL repone al computador a su estado preinterrupción y hace que reasuma su procesamiento de programa principal.

Si Ud. no tiene ganas de implementar por sí mismo un programa de este tipo, hay uno disponible. Hay un editor BASIC que permite la creación y modificación de datos de sonido a medida que se escuchan. Eso se consigue por medio de un generador de sonido de interrupción tal como se indicó mas arriba: comandado por tabla y compatible con todos los lenguajes. El paquete se llama INSOMNIA (Interrupt Sound Initializer/Altered), y se ofrece a través del intercambio de programas ATARI.

GENERACION DE SONIDO CON CODIGO DE MAQUINA: El usar lenguaje assembler abre nuevas puertas a la generación de sonido. Se pueden hacer intentos ahora de simular instrumentos musicales específicos. La técnica consiste en lo siguiente: escriba un programa que se parezca a la rutina de interrupción de 60 Hz, en que es comandada por una tabla. La salida de esta rutina para 3 notas musicales tendrá una apariencia como la siguiente:

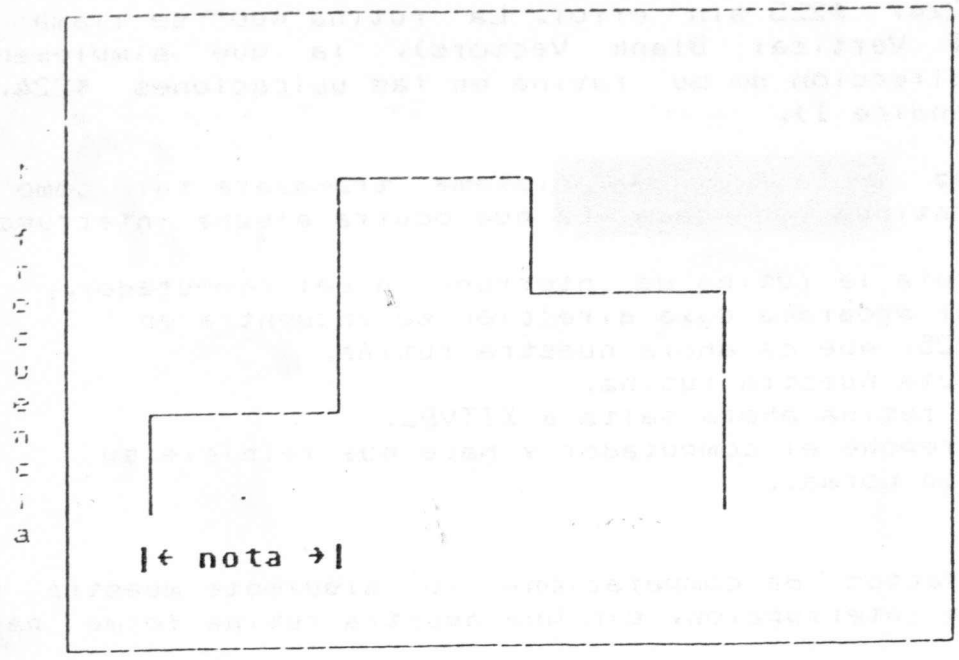


Figura 10.12
Notas musicales tocadas en una rutina musical normal

Como tenemos mucho tiempo de procesamiento disponible, podemos llegar a un nivel más profundo, alterando levemente la frecuencia durante el tiempo de ejecución de la nota, de modo que parezca un instrumento. Por ejemplo, supongamos que hemos descubierto que podemos obtener un sonido idéntico al que se produce cada vez que se golpea la tecla de un piano (cualquier tecla), interpretando muy rapidamente una tabla de frecuencias. Esta tabla podría tener la siguiente apariencia:

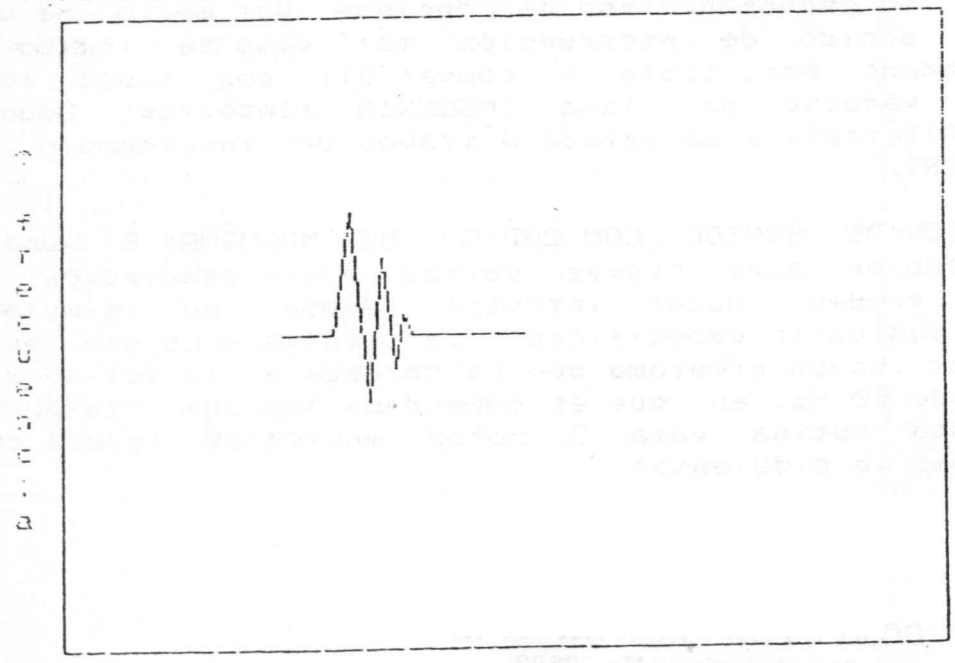


Figura 10.13
Gráfico de las frecuencias que se dan en una nota de piano

Llamemos la tabla de más arriba la "tabla de envolvente" y sus datos "las envolventes de piano". Para simular un piano, la idea sería sumar rápidamente la envolvente de piano a la nota básica. Así la nota se modifica levemente durante su tiempo de ejecución. Por ejemplo, una simulación de piano para las 3 notas de la figura 3.12 se vería así:

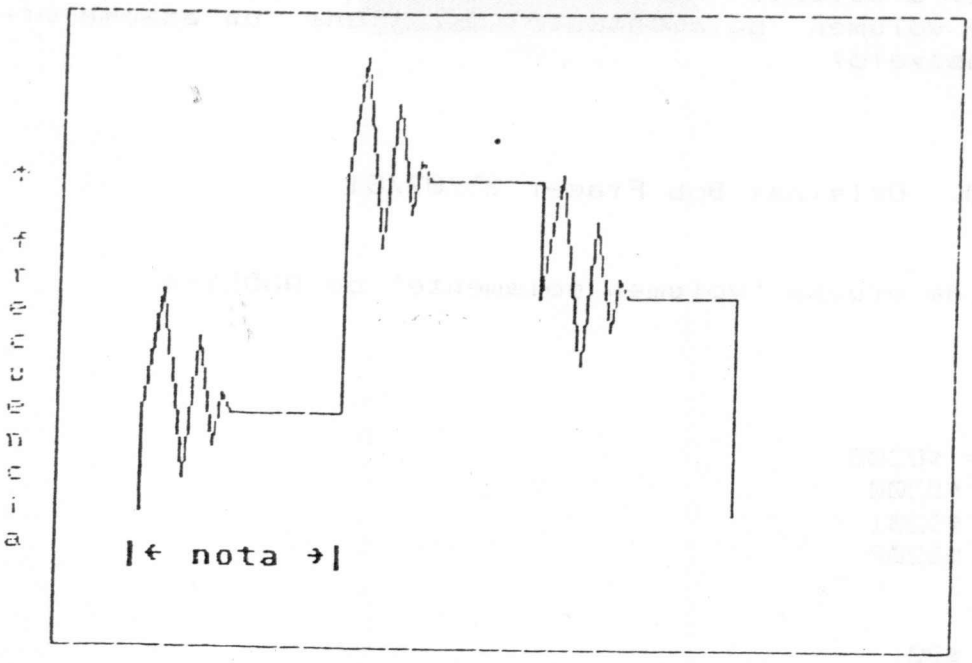


Figura 10.14
El ejemplo de las 3 notas de la Figura 10.12 tocadas con una envolvente de piano

Tenemos ahora esencialmente el mismo sonido producido por la rutina de música normal de la Figura 10.12- solamente que ahora las notas tienen un sabor a piano y un sonido mucho más agradable que unos simples "pitos".

Desafortunadamente hemos tenido que sacrificar todo otro procesamiento para obtener esta apariencia de piano. El sonido ya no se revisa una vez por nota solamente , sino que tal vez unas 100 veces dentro de la duración de cada nota.

Volumen solamente: Anteriormente habíamos experimentando con los bits de volumen solamente de AUDC1-4, e indicado una potencia escondida, descubriendo también que aparentemente no tenía mucho uso. Esto se debió totalmente al hecho de que BASIC es demasiado lento para hacer un uso efectivo. ¡No es así con lenguaje de máquina.

Como dijimos antes: este bit ofrece una gran capacidad para una reproducción precisa del sonido. La generación de las verdaderas formas de ondas (hasta los límites de resolución de volumen y tiempo del computador), ahora es posible. En vez de simplemente poner un sabor a piano en la música, podemos

duplicar ahora el sonido del piano.

Desafortunadamente, nunca podrá simular exactamente un instrumento. 4 bits (15 valores) no dan suficiente resolución en volumen, aunque de ninguna manera esta técnica es poco fructífera. El siguiente programa muestra el uso de uno de estos bits de volumen solamente. Si Ud. tiene un assembler, digítelo y ensáyelo:

```
0100 ;
0110 ; SOLOVOL  Original Bob Fraser 23/07/81
0120 ;
0130 ;
0140 ;rutina de prueba 'volumen solamente' de AUDC1-4
0150 ;
0160 ;
0170 ;
0180 ;
0190 AUDCTL = $D208
0200 AUDF1 = $D200
0210 AUDC1 = $D201
0220 SKCTL = $D20F
0230 ;
0240 ;
0250      *=  $80
0260 RITMO .BYTE 1
0270 CMS .BYTE 0
0280 ;
0290 ;
0300 ;
0310 *=$4000
0320      LDA #0
0330      STA AUDCTL
0340      LDA #3
0350      STA SKCTL
0360      LDX #0
0370 ;
0380      LDA #0
0390      STA $D40E   eliminar VBI (interr. borr. vert.)
0400      STA $D2E0   eliminar IRQ (Interrupciones enmascarables)
0410      STA $D400   eliminar DMA (Acceso directo de memoria)
0420 ;
0430 ;
0440 ;
0450 HOLA LDA TABD,X
0460      STA  CMS
0470 ;
0480      LDA TABV,X
0490 L0   LDY RITMO
0500      STA AUDC1
0510 L1   DEY
0520      BNE L1
```

```
0530 ;
0540 ;Disminuir control mas signif.
0550      DEC CMS
0560      BNE L0
0570 ;
0580 ;
0590 ; nota nueva
0600 ;
0610      INX
0620      CPX CN
0630      BNE HOLA
0640 ;
0650 ;rotar puntero de notas
0660      LDX #0
0670      BEQ HOLA
0680 ;
0690 ;
0700 CN   .BYTE 28     contador de notas
0710 ;
0720 ;tabla de los volumenes a tocarse sucesivamente
0730 TABV
0740      .BYTE 24,25,26,27,28,29,30,31
0750      .BYTE 30,29,28,27,26,25,24
0760      .BYTE 23,22,21,20,19,18,17
0770      .BYTE 18,19,20,21,22,23
0780 ;
0790 ;esta tabla contiene la duracion de c/u de la notas de arriba
0800 TABD
0810      .BYTE 1,1,1,2,2,2,3,6
0820      .BYTE 3,2,2,2,1,1,1
0830      .BYTE 1,1,2,2,2,3,6
0840      .BYTE 3,2,2,2,1,1
```

Sorprendentemente, la velocidad no constituye aquí un problema. La onda tiene casi 50 pasos y el prosrama aún funciona con una onda muy aguda (aprox. 10KHz).

Elimine las líneas 400-410 y vuelva a ensayar el programa. Sonará bastante entrecortado. La causa se encuentra en las interrupciones de 60 hz discutidas en la sección anterior. De hecho pueden escucharse las interrupciones ya que todo el sonido se detiene durante ese tiempo.

La línea 420 inhibe el acceso directo de memoria (DMA) de la pantalla. Por ello la pantalla toma el color de fondo mientras se ejecuta el programa. El propósito es doble: acelerar el procesador y hacer consistente la sincronización, ya que el DMA toma ciclos a intervalos irregulares. Vea el capítulo 5 para mayorrs antecedentes acerca del DMA.

En este caso particular, el sonido creado es una onda sinusoidal. La onda es notablemente pura, y realmente suena como una onda sinusoidal. Al graficarlos, los datos se ven así:

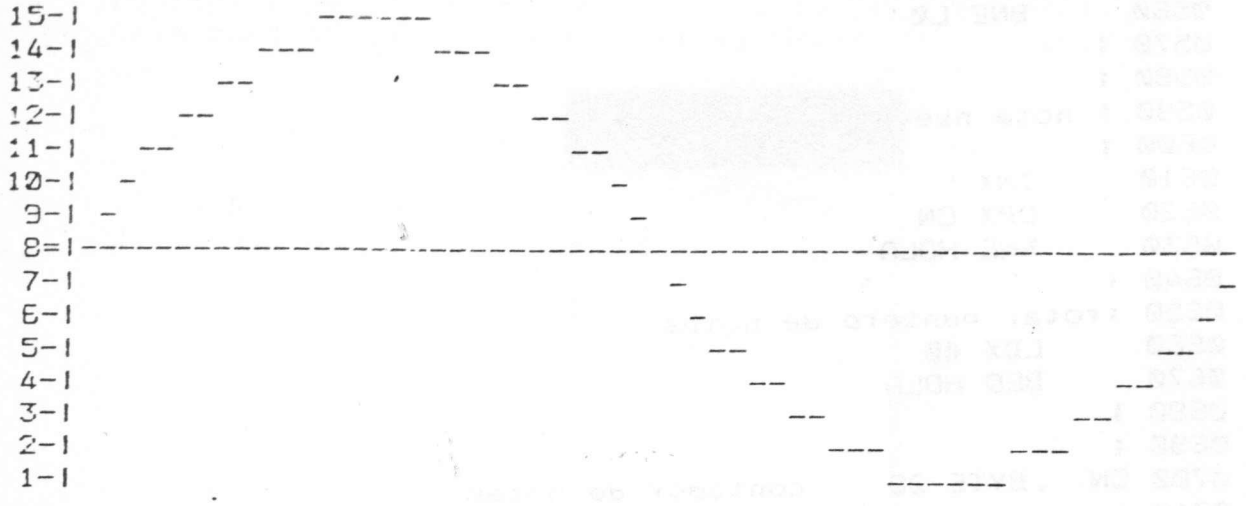


Figura 10.15
Graficación de los datos de onda sinusoidal para el programa precedente de volúmen solamente

Puede lograrse mucho con las capacidades de sonido ATARI. La cuestión es: porqué sonido?

Los productores de películas hace mucho que captan la importancia de la música de fondo creadora de ambientes. En las últimas películas de aventuras espaciales de George Lukas tenemos ejemplos excelentes. Cuando el villano entra a la sala se sabe inmediatamente que hay que temerlo y odiarlo, debido a los ritmos de fondo amenazantes que acompañan su entrada. Y aplaudiremos con alivio cuando el héroe salve a la princesa, mientras entre bambalinas se escuche música galante. Las películas de horror pueden asustarlo a uno simplemente tocando música fantasmal, a pesar de que la acción que transcurra pueda ser absolutamente corriente.

SPACE INVADERS crea una tensión personal a sus jugadores y víctimas con sus melodías de eco. A medida que el ritmo aumenta, los nudillos se ponen blancos y los dientes crujen. Cuando un Zylon de Star Riders dispara un torpedo de fotones, Ud. tira frenéticamente el control para evitar el impacto. A medida que se dirige derecho hacia Ud., el tiempo transcurre en forma más lenta y Ud. escucha su silbido más y más fuerte a medida que se acerca. Justo antes del impacto Ud. se agacha y desaparece de su sillón.

Los sonidos impresionistas afectan nuestro subconsiente y nuestro estado de ánimo. Esto posiblemente se deba al hecho de que los sonidos, si es que están presentes, entran en forma continuada a nuestra mente, estemos o no escuchando concientemente. Por otro lado, si nos distraemos del

televisor, cesamos de concentrarnos en la imagen y ella abandona nuestra mente. Por ello los sonidos ofrecen al programador un camino directo hacia la mente del usuario - cruzando su proceso de pensamiento y apuntando directamente hacia sus emociones.

Incluso un juego aburrido puede volverse excitante con sonidos ambientales de fondo, jubilosos sonidos de éxito y frustrantes pero esperanzados sonidos de fracaso. Esto puede hacer que un juego sea mucho más popular, aunque tome también un gran esfuerzo en su desarrollo.

APENDICE I
INTERRUPCIONES DEL INTERVALO VERTICAL

Los computadores ATARI poseen una serie de interrupciones que pueden ser de gran utilidad. Este apéndice cubrirá las interrupciones del intervalo de borrado vertical. Son éstas, interrupciones no enmascarables que ocurren cada 60avo de segundo, durante el período de borrado vertical del despliegue de televisión. Tienen una amplia variedad de usos.

Al comienzo del borrado vertical, ANTIC baja la línea NMI (interrupción no enmascarable, Non Maskable Interrupt) del 6502. El 6502 a continuación vectoriza a una rutina de servicio de interrupción no enmascarable, que determina la fuente de la interrupción. Si se trata de una interrupción de borrado vertical, el 6502 empuja sus registros A, X e Y al stack y salta a través del vector de borrado vertical inmediato (VVBLKI) ubicado en $0222. Este vector normalmanete apunta a una rutina de servicio de interrupción de borrado vertical del sistema operativo en $E45F. Esta rutina finaliza saltando a través del vector de interrupción del borrado vertical diferido (VVBLKD) en $0224. Normalmente este vector apunta a la rutina de terminación de interrupción simple en $E462. La Figura I.1 ilustra este proceso.

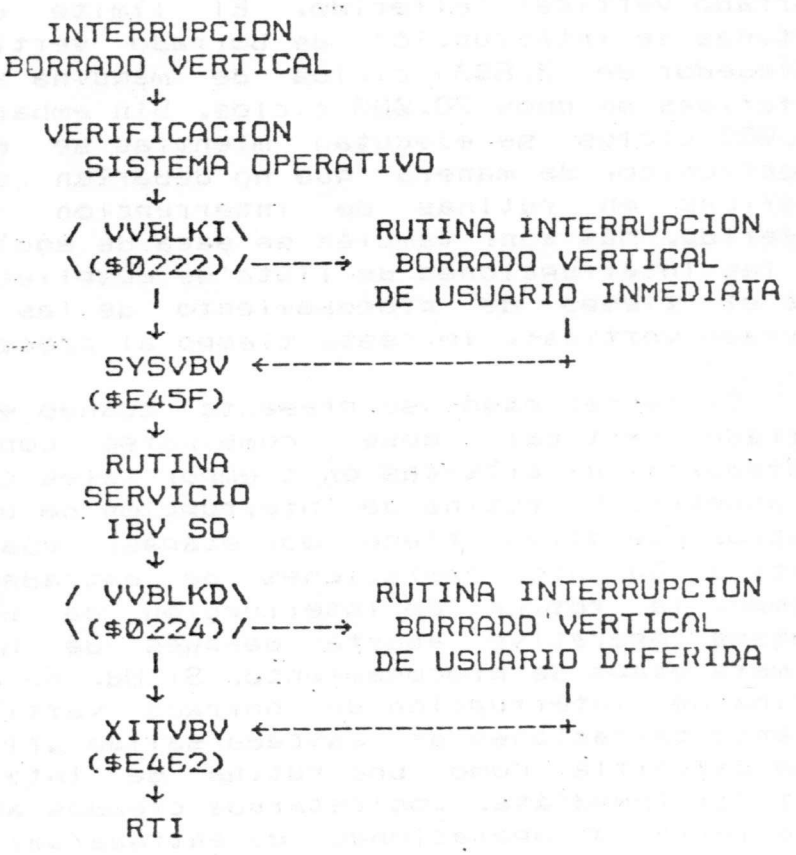


Figura I.1
Ejecución normal de una interrupción de borrado vertical
(y cómo modificarla)

Estos dos vectores se encuentran en RAM para que el programador pueda atrapar las rutinas de servicio de interrupción y usar la interrupción de 60 Hertz para sus propios propósitos. El procedimiento para usarlas es muy simple. Primero decida si la rutina de interrupción de borrado vertical (VBI) será un VBI inmediato o un VBI diferido. En muchos casos hay poca diferencia en cual se elige. Solamente existen unas pocas ocasiones en que importa. El primer caso se presenta cuando su rutina VBI lee o escribe a registros que son copiados por la rutina de interrupción de borrado vertical del sistema operativo. Por ejemplo puede ser necesario escribir a los registros circuitales después que la rutina VBI del sistema operativo lo haya hecho, para tener, por decirlo así, la última palabra.

El segundo caso se presenta cuando su rutina de interrupción de borrado vertical consume demasiado tiempo del procesador. En este caso, la rutina de interrupción de borrado vertical del sistema operativo puede verse demorada más allá del final del período de borrado vertical. En este caso, es posible que algunos registros gráficos sean modificados mientras el haz esté trazando en la pantalla. El resultado puede ser desagradable. Si éste es el caso, su rutina de interrupción de borrado vertical debe definirse como una de borrado vertical diferido. El límite de tiempo para las rutinas de interrupción de borrado vertical inmediato es de alrededor de 3.800 ciclos de máquina y para las rutinas diferidas de unos 20.000 ciclos. Sin embargo, muchos de estos 20.000 ciclos se ejecutan mientras se está trazando el haz electrónico, de manera que no deberían ejecutarse operaciones gráficas en rutinas de interrupción de borrado vertical diferida. Más aún, también se saca de aquí el tiempo de ejecución de las interrupciones de lista de despliegue. Recuerde también que el tiempo de procesamiento de las interrupciones de borrado vertical, le resta tiempo al procesamiento principal.

El tercer caso se presenta cuando esta interrupción de borrado vertical debe combinarse con operaciones de entrada/salida críticas en tiempo, tales como acceso al disco o cassette. La rutina de interrupción de borrado vertical del sistema operativo tiene dos etapas, una crítica y una no crítica. Durante operaciones de entrada/salida críticas en tiempo, la rutina de interrupción de borrado vertical del sistema operativo aborta después de haberse cumplido la primera etapa de procesamiento. Si Ud. no desea que su propia rutina de interrupción de borrado vertical sea desactivada durante operaciones de entrada/salida críticas en el tiempo, debe definirla como una rutina de interrupción de borrado vertical inmediata. Los retardos creados así, a su vez pueden interferir con operaciones de entrada/salida críticas en el tiempo. Este será un nuevo problema para Ud..

Una vez que Ud. haya decidido si su rutina de interrupción de borrado vertical es del tipo inmediato o

diferido, debe ponerla en memoria (la página 6 es un excelente lugar), enlazar su finalización con el procesamiento regular de las interrupciones de borrado vertical y modificar el vector RAM del sistema operativo correspondiente para apuntar a ella. Termine una rutina de interrupción de borrado vertical inmediata con un JMP a $E45F. Finalice una rutina de interrupción de borrado vertical diferida con un JMP a $E462. Si Ud. quiere anular completamente la rutina de interrupción del borrado vertical del sistema operativo (y así ahorrar algún tiempo de procesamiento), finalice su rutina de interrupción de borrado vertical inmediato con un JMP a $E462.

Un problema común de las interrupciones en los microcomputadores de 8 bits se presenta al tratar de cambiar el vector para la interrupción. Los vectores son cantidades de dos bytes; se requieren dos instrucciones para su cambio. Existe una pequeña probabilidad de que una interrupción se presente una vez que se haya modificado el primer byte, pero antes de hacerlo con el segundo. Esto bloquearía el sistema. La solución para este problema la da una rutina del sistema operativo llamada SETVBV en la ubicación $E45C. Cargue el registro Y del 6502 con el byte menor de la dirección y el registro X con el byte mayor de la dirección, ponga en el acumulador un 6 para el caso de una interrupción inmediata y un 7 para una interrupción diferida. Enseguida haga un JSR SETVBV y la interrupción se habilitará en buena forma. Comenzará ejecutando dentro de un 60avo de segundo.

Puede realizarse una gran variedad de operaciones con interrupciones de 60 Hertz. Primero, las manipulaciones de pantalla se pueden hacer durente el borrado vertical, para asegurar que las transiciones no ocurran sobre la pantalla, en segundo lugar pueden realizarse manipulaciones regulares sobre pantallas de alta velocidad, esto es importante al mostrar muchos tipos de animación. Por ejemplo, las burbujas del programa de reactor nuclear de SCRAM (TM) deben moverse a un ritmo regular. No deben acelararse ni frenarse a medida que otras actividades computacionales se cargan sobre el 6502. La única forma de asegurar esta regularidad de movimiento es realizar la animación durante las interrupciones del borrado vertical.

Otra función de las interrupciones de borrado vertical se encuentra en la generación de envolventes de sonido. Los registros de sonido de los computadores ATARI permiten el control de frecuencia, volumen y distorsion, no así de la duración. La duración puede controlarse con una interrupción de borrado vertical, haciendo que la rutina de sonido disponga un parámetro de duración. A continuación, la rutina de interrupción de borrado vertical puede ir disminuyendo este parámetro hasta que llegue a 0 y apagar el sonido en ese momento. Esta técnica puede usarse igualmente para controlar el volumen del sonido y dar envolventes de ataque y caída a éstos. El control de la frecuencia y de la distorsión es

posible con versiones extendidas de esta técnica. Esta técnica puede producir efectos sonoros muy intrincados, debido a que la resolución de tiempo es de un 60avo de segundo solamente. Las interrupciones de borrado vertical no son muy útiles para el control directo de la amplitud del parlante.

Las interrupciones de borrado vertical son útiles también para manejar entradas de usuario. Estas entradas generalmente requieren pocos cálculos, pero una atención constante. Una interrupción de borrado vertical permite que el programa verifique si existe una entrada de usuario, cada 60avo de segundo, sin sobrecargar en otra forma el programa. Es ésta una solución ideal para el problema de mantener la continuidad de la computación sin ignorar al usuario.

Por último, las interrupciones de borrado vertical proveen una forma cruda de multitarea. Un programa frontal puede hacerse funcionar bajo el modo de interrupciones de borrado vertical, mientras un programa de fondo corre en la línea principal. Como con cualquier interrupción, debe darse un cuidado especial a la separación de las bases de datos de ambos programas. Sin embargo, la potencia lograda, bien puede valer el esfuerzo.

APENDICE II
INGENIERIA HUMANA

El Sistema de Computación Personal ATARI es primero y ante todo un computador para el usuario general. La circuiterìa se diseñó en tal forma para que este computador sea fácil de usar. Posee muchas caracterìsticas circuitales que protegen al usuario de cometer errores inadvertidamente. Los programas escritos para este computador deberìan reflejar igual preocupación por la impericia del usuario. El consumidor promedio no es estúpido, simplemente no está familiarizado con las convenciones y tradiciones del mundo de la computación. Una vez que entiende un programa lo usará en buena forma la mayorìa de las veces. Ocasionalmente será descuidado y hará errores. Es la responsabilidad del programador el proteger al usuario de sus propios errores.

El estado actual de la ingenierìa humana de programación en la industria de los computadores personales es realmente un desastre. Se ofrece una gran cantidad de programas que poseen una ingenierìa humana exiguamente pobre. Los más ofensivos son los programas escritos por aficionados, pero incluso los escritos por alguna de las grandes firmas muestran ocasionalmente lapsos en la ingenierìa humana.

La ingenierìa humana es un arte, no una ciencia. Requiere de gran pericia técnica, pero también conocimiento y sentimiento. Como tal es un campo altamente subjetivo, carente de referencias absolutas. Este apéndice es la labor de una mano y asì refleja la subjetividad de su autor. Una perspectiva apropiada que considerará la amplia variedad de opiniones sobre esta materia habrìa inflado el apéndice más allá de un lìmite razonable de extensión. Mas aún, una presentación completa de todos los puntos de vista solamente confundirìa al usuario con sus muchas aseveraciones, calificaciones, contraposiciones y contradicciones. Por ello he escogido la tares más simple de presentar solamente mi punto de vista, mencionando en forma breve las objeciones más serias. El resultado es suficientemente contradictorio para satisfacer aún al más académico de los lectores.

EL COMPUTADOR COMO SER CONSCIENTE

Una forma instructiva de mirar el problema de la ingenierìa humana es considerar al programador como un exorcista conjurando un ser inteligente, un homúnculo que se encuentra en las entrañas del computador. Esta creación carece de cuerpo pero posee rasgos intelectuales, especificamente la habilidad de procesar y organizar información. El usuario del programa entra a una relación con su homúnculo. Los dos seres conscientes piensan en forma diferente; los rasgos del pensamiento humano son asociativos, integrados y difusos, mientras los procesos mentales del programa son directos, analìticos y especìficos. Estas diferencias son complementarias

y productivas, ya que el homúnculo puede hacer tan bien lo que el humano no puede. Desafortunadamente estas diferencias también crean una barrera de comunicación entre el humano y el homúnculo. Tienen tanto que decirse uno al otro por ser tan diferentes, pero porque son diferentes no pueden comunicarse bien. El problema central de una buena programación debe ser por lo tanto el proveer una buena comunicación entre el usuario y el homúnculo. Es triste tener que decirlo pero muchos programadores gastan la mayor parte de sus esfuerzos aumentando y mejorando la potencia de procesamiento de sus programas. Esto solamente tiene como resultado un ser más inteligente pero sin ojos para ver ni boca para hablar.

La actual cosecha de computadores personales ha logrado metas que los hacen capaces de soportar programas de suficiente inteligencia para satisfacer muchas necesidades del consumidor promedio. El principal factor limitante ya no es velocidad del reloj o la memoria residente; el principal factor limitante es ahora el angosto canal que conecta nuestro homúnculo ya inteligente con su usuario humano. Cada uno de ellos puede procesar la información en forma rápida e inteligente; solamente este angosto canal entre ambos reduce la interacción.

LA COMUNICACION HOMBRE MAQUINA

Como podremos aumentar la capacidad del canal entre los dos pensadores? Debemos enfocar el lenguaje en que se comunican. Como calquier lenguaje, un lenguaje hombre-máquina está restringido por los medios físicos de expresión que se encuentran disponibles para los interlocutores. Como el computador y el hombre son físicamente diferentes, sus modos de expresion también lo son. Esto obliga a usar un lenguaje que no sea bidireccional (como lo son los lenguajes humanos). En vez de ello un lenguaje hombre-máquina tendrá dos canales, un canal de entrada y un canal de salida. Tal como estudiamos un lenguaje humano estudiando primero los sonidos que las cuerdas vocales humanas pueden generar, comenzamos aquí examinando los componentes físicos de la interfaz hombre-máquina.

LA SALIDA (DEL COMPUTADOR HACIA EL HUMANO)

Hay fundamentalmente dos canales de salida desde el computador hacia el usuario. El primero es la pantalla del televisor, el segundo el parlante. Afortunadamente son dispositivos flexibles que permiten un amplio rango de expresión. El cuerpo principal de este libro describe las posibilidades disponibles desde el punto de vista del computador. Para los propósitos de este apéndice es mas útil discutir estos dispositivos en términos del punto de vista humano. De los dos dispositivos (pantalla y parlante) la pantalla de despliegue se entiende fácilmente como el dispositivo más expresivo y poderoso. El ojo humano es un dispositivo de captación de información desarrollado en forma más fina de lo que es el oído humano. En términos de ingeniería eléctrica, tiene más ancho de banda que el oído. El ojo puede procesar tres modalidades principales de información visual: forma, color y movimiento.

Formas

Las formas son un medio ideal para presentar información al humano. La retina humana está especialmente adaptada al reconocimiento de formas. El uso más directo de las formas es para la representación directa de objetos. Si Ud. quiere que el programa ilustre al usuario acerca de algo, trace una figura de ese algo. Una imagen es algo directo, obvio e inmediato.

El segundo uso de las formas es para símbolos. Algunos conceptos del léxico humano no tienen una imagen directa, Conceptos como amor, infinito y dirección no pueden representarse por imagen. Deben en cambio transmitirse por medio de símbolos, tales como un corazón, un 8 en posición horizontal o una flecha. Son éstos, algunos de los muchos símbolos que todos reconocemos y usamos. A veces se podrá crear un símbolo ad hoc para uso limitado en el programa. La mayoría de la gente será capaz de adaptar un símbolo ad hoc como éste en forma bastante fácil. Los símbolos constituyen una forma compacta de expresar una idea, pero no deberían ser usados en lugar de imágenes a no ser que su carácter compacto sea esencial. Un símbolo es una expresión indirecta. Una imagen directa. La imagen transmite la idea con mayor fuerza.

El tercero y mas común de los usos de las formas es para texto. Una letra es un símbolo; ponemos letras una al lado de la otra para formar palabras. El lenguaje así producido es extremadamenre rico en su poder expresivo. Es muy cierta la expresión "si no puede decirlo, Ud. no puede conocerlo". Este poder expresivo se gana a un cierto precio: la indirección extrema. La palabra que expresa una idea no tiene una conexión sensitiva o emocional con la idea. El ser humano debe

realizar una extensa gimnasia mental para decifrar la palabra. Naturalmente, lo hacemos con tal frecuencia que hemos alcanzado una gran fluidez en la traducción de cadenas de letras a ideas. No somos conscientes del esfuerzo. El punto importante está en que la indirección reduce el carácter inmediato y la fuerza de la comunicación.

Hay una escuela de pensamiento que mantiene que el texto es superior a la gráfica para propósitos de comunicación. La esencia del argumento es que el texto anima al lector para hacer un uso más libre de su imaginación. Este argumento no me satisface porque es el lector el que debe usar su imaginación, está aportando información que no estaba inherente en la comunicación propiamente tal. El mismo ejercicio de imaginación hecho con gráfica daría resultados aún mayores. Un argumento más convincente a favor del texto es que su indirección permite apretar una gran cantidad de información dentro de un espacio pequeño. Las restricciones de espacio en una comuniación hacen valioso lo compacto de un texto. Sin ambargo, esto no convierte al texto en algo superior a la gráfica, lo hace más económico. La gráfica requiere más espacio, más tiempo, memoria o dinero. Pero igualmente comunica mejor que el texto. Hasta cierto punto la elección entre gráfica y texto es una materia de gusto y el gusto del público comprador esta más allá de este punto. Compare la popularidad de la televisión con la de la radio, de las películas o de los libros. Fácilmente la gráfica bate al texto.

Color

El color es otro vehículo para llevar información. Es menos poderoso que la forma, de manera que normalmente juega un rol secundario a la forma en las presentaciones visuales. Su uso más frecuente es para diferenciar entre formas que de otras formas serían iguales. También juega un rol importante aportando índices al usuario. Un buen color puede salvar una forma que sin él sería ambigua; por ejemplo, un árbol representado como carácter debe caber dentro de una red de pixel de 8x8. La red es demasiado pequeña para trazar un árbol reconocido, sin embargo, coloreando el árbol de verde, la imagen llega a ser mucho más fácil de reconocer. El color también es útil para atraer la atención o para señalizar algún material importante. Colores cálidos atraen la atención. El color también dá una mejor estética. Las imágenes coloreadas son más agradables de mirar que las que simplemente están en blanco y negro.

Animación

Uso el término "animación" aquí para describir un cambio visual. La animación incluye el cambio de colores, el cambio de formas, el movimiento de objetos en el primer plano o el movimiento del fondo. El valor principal de la animación se encuentra al mostrar procesos dinámicos. De hecho la animación

gráfica es la única forma para presentar exitosamente acontecimientos altamente activos. El valor de la animación se demuestra con mayor intensidad en juegos como STAR RAIDERS. Es imposible imaginarse cómo sería este juego sin animación. A propósito, podría Ud. imaginar lo que sería si sólo tuviera texto? El valor de la animación se extiende más allá de los juegos. La animación permite al di)522ador mostrar claramente acontecimientos dinámicos, cambiantes. La animación es una de las mayores ventajas que los computadores tienen sobre el papel en la tecnología de la información. Por último, la animación es muy poderosa en términos sensoriales. El ojo humano está organizado para responder fuertemente a los cambios del campo visual. La animación puede atraer la atención del ojo y aumentar el compromiso del usuario con el programa.

Sonido

Es necesario mirar las imagenes gráficas para que tengan algún efecto. El sonido puede llegar al usuario aún si el usuario no está poniendo atención directa al sonido. Por ello el sonido tiene gran valor como anunciador o índice de advertencia. Una gran variedad de pitos, tonos y gruñidos pueden se022alizar realimentación al usuario. Las acciones correctas pueden ser respondidas con agradable tono de campana. Acciones incorrectas pueden contestarse con carraspera. Las condiciones de advertencia pueden hacerse notar con un bocinazo.

El sonido también tiene una segunda aplicación: proveer afectos sonoros realistas. Los efectos sonoros de buena calidad pueden agregar mucho al impacto de un programa, porque el sonido provee un segundo canal de información que es muy efectivo, incluso cuando el usuario tiene la vista ocupada.

El sonido se presta poco para la comunicación de informaciones exactas; la mayoría de la gente no tiene la acuciosidad acústica para distinguir diferencias pequeñas entre tonos. El sonido es mucho mas efectivo para transmitir estados emocionales o respuestas. La mayoría de la gente tiene una multitud de asociaciones de sonidos con estados emocionales. Una secuencia descendente de notas implica circunstancias que se están deteriorando. El sonido de una explosión indica destrucción. Una fanfarria auncia una llegada importante. Algunas secuencias de notas de canciones populares muy conocidas se asocian inmediatamente con sentimientos particulares. Por ejemplo, en ENERGY CZAR", usé una melodía de funeral para indicar al usuario que su mal manejo de energía había arruinado la situación energética de América, y un fragmento de "Happy Days Here Again" para indicar éxito.

DISPOSITIVOS DE ENTRADA (DEL HUMANO AL COMPUTADOR)

Son tres los dispositivos de entrada más comúnmente usados con el Sistema de Computación Personal ATARI. Ellos son teclado, bastón y paleta.

Teclado

El teclado se entiende fácilmente como el dispositivo de entrada más poderoso disponible al diseñador. Tiene más de cincuenta teclas directas de acceso inmediato. El uso de las teclas CONTROL y SHYFT más que duplican el número de entradas diferentes que el usuario puede emplear. Las teclas ATARI o video inverso y mayúscula/ minúscula extienden el rango expresivo del teclado aún más allá. Así, con un simple toque de tecla el usuario puede indicar cualquiera de 125 comandos. Un par de teclas pueden definir más de 15.000 selecciones. Obviamente, es un dispositivo sumamente expresivo; fácilmente puede hacerse cargo de los requerimientos de comunicación de cualquier programa. Por esta razón el teclado es el dispositivo de entrada elegido por la mayoría de los programadores.

Aunque la potencia del teclado es innegable, sus debilidades muchas veces no se reconocen. La primera debilidad es que simplemente mucha gente no sabe usarlo bien. Los programadores lo usan intensamente en su trabajo diario; en consecuencia son digitadores rápidos. El consumidor promedio no se siente tan bien frente a un teclado. Le es fácil equivocarse de tecla. La sola existencia de todas esas teclas y el saber que uno debe tocar la tecla correcta de por sí ya resulta intimidante para mucha gente.

Una segunda debilidad del teclado es su indirección. Es muy difícil dar un significado directo a un teclado. El teclado no tiene un significado sensorial o emocional definido. El usuario nuevo tiene grandes dificultades de conectarse con él. Todo el trabajo de teclado es simbólico, en él se usan teclas marcadas con símbolos, que a su vez tienen el significado según las circunstancias. La indirección de todo esto puede llegar a confundir mucho al novicio. Los teclados también adolecen de su natural asociación con los despliegues de texto; ya discutí las debilidades del texto como medio de transferencia de información.

Otra propiedad del teclado que el diseñador debe mantener in mente es su carácter digital. El teclado es digital tanto en su selección como en el tiempo. Esto naturalmente da una protección contra errores. Como la lectura de teclado a través del tiempo no es un proceso continuo, sino digital el teclado no se presta bien para apliaciones en tiempo real. Como los seres humanos son creaciones de tiempo real, esto constituye una debilidad. El diseñador debe tener claro, que el uso del teclado siempre lo alejará de una interacción en tiempo real con su usuario.

Paletas

Las paletas son los únicos dispositivos de entrada verdaderamente analógicos que existen para el sistema. Como tales adolecen del problema normal de todos los dispositivos de entrada analógicos: el requerimiento de que el usuario haga ajustes precisos para llegar a un resultado. Su resolución angular es pobre y el efecto térmico produce algún temblor aún en la salida de una paleta que no está siendo tocada.

Su principal valor es doble. Primero, son muy apropiadas para escoger valores de una variables unidimensional. La gente de inmediato toma la idea que la paleta barre a través de todos los valores y el presionar el disparador comunica la selección. En segundo lugar, el usuario puede barrer de un extremo del espectro a otro con el simple giro de un dial. Esto hace que todo el espectro de valores sea de acceso inmediato al usuario.

Un factor importante en el uso de las paletas es la creación de un bucle cerrado entrada/salida. En la mayoría de los procesos de entrada es de desear que se produzca un eco de las entradas hacia la pantalla de modo que el usuario pueda verificar el ingreso que haya producido. Este proceso de eco crea un bucle cerrado de entrada/salida. La información viaja del usuario, al dispositivo de entrada, al computador, a la pantalla y al usuario. Como las paletas no tienen posición absoluta, este eco es esencial.

Cualquier conjunto de entradas que con algún sentido pueda ponerse a lo largo de una secuencia lineal, puede accederse con una paleta. Por ejemplo, los menús pueden direccionarse con una paleta. La secuencia es de arriba - abajo en el menú. Es muy posible (pero nada razonable) sustituir el teclado por una paleta. La paleta barre a través de las letras del alfabeto, mostrando en la pantalla la letra escogida en cada momento. Presionando el disparador de la paleta se selecciona la letra. Si bien este esquema no será capaz de producir records de rapidez de digitación, es útil para niños y la idea puede aplicarse a otros problemas.

Bastones

Los bastones son los dispositivos de entrada mas simples que existen para un computador. Son muy firmes y por lo tanto pueden usarse en ambientes ásperos. Contienen solamente 5 conmutadores. Por esta razón muchas veces se subestima su poder expresivo. Sin embargo, los bastones son sorprendentemente útiles como dispositivos de entrada. Al usarlo como impulsor, un bastón puede direccionar cualquier punto sobre la pantalla, haciendo otra vez la selección con el botón rojo. Con una buena disposición de pantalla, el bastón puede dar así una gran variedad de funciones de control. Recurrì a un bastón para controlar un reactor nuclear (SCRAM") y para la ejecución de un juego de guerra (FRENTE ORIENTAL 1941).

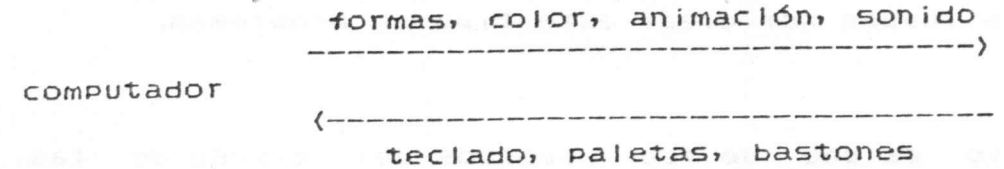
La clave para el buen uso de un bastón está en darse cuenta que la variable crìtica no es tanto la selección del conmutador, sino el tiempo durante el cual el conmutador se mantiene presionado. Controlando esta duración de tiempo, el usuario determina cuanto debe movese el cursor. Para ello, normalmente se requiere de un cursor de velocidad constante. Un cursor de velocidad constante introduce un compromiso difìcil. Si el cursor se mueve demasiado rápido, el usuario tendrá dificultades para ubicarlo en el item de su selección. Si el cursor se mueve en forma demasiado lenta, el usuario se volverá impaciente, esperando que atraviese distancias grandes a través de la pantalla. Una solución a este problema es el cursor con aceleración. Si el cursor comienza a movese lentamente y despues se acelera, el usuario puede tener tanto posicionamiento fino como alta velocidad.

El verdadero valor de un bastón está en su alto grado de tacto. El bastón compromete al usuario en sus ingresos a través de una vìa sensorial y directa. EL tacto en el teclado no tiene un significado emocional, en cambio en un bastón tiene sentido. Empujar hacia arriba para subir, hacia abajo para bajar. Si el cursor refleja esto en la pantalla, todo el proceso de entrada tiene mucho más sentido para el usuario.

También los bastones tienen sus limtaciones. A pesar de que es posible presionar el bastón en posición diagonal y obtener una lectura correcta de esa dirección, estas direcciones no son suficientemente definidas como para permitir ingresos diagonales como comandos separados. Tal como algunas palabras (por ejemplo "prestidigitador", "vetrìlocuo") son difìciles de enunciar claramente, así también las órdenes diagonales son difìciles de ingresar en forma clara. Así, deberìan evitarse los valores diagonales, a no ser que se usen en el sentido estrictamente geométrico: empujar hacia arriba el bastón significa hacia arriba, hacia la derecha significa derecha y en diagonal significa diagonalmente.

RESUMEN DE LOS ELEMENTOS DE LA COMUNICACION

He discutido una serie de caracterìsticas y dispositivos que en conjunto constituyen los elementos de un lenguaje para la interacción del computador y el usuario. Ellos son:



CONSTRUYENDO UN LENGUAJE

Cómo armamos todos estos elementos para formar un lenguaje eficaz? Para hacerlo debemos, en primer lugar, determinar los rasgos principales que esperamos de un buen lenguaje. Ellos son:

Integridad

El lenguaje debe expresar totalmente las ideas que deben comunicarse entre el computador y el usuario. No tiene por qué expresar ideas que sean internas a los procesos de cualquiera de los dos pensadores. Por ejemplo, el lenguaje empleado en STAR RAIDERS debe expresar todos los conceptos relacionados con el control de la nave y con la situación del combate. No es necesario que exprese la ansiedad del jugador o las intenciones de trayectoria de vuelo de los Zylones. Estos conceptos, si bien fundamentales para la función de todo el juego, no necesitan comunicarse entre el usuario y el computador.

La integrida es una función obvia de cualquier lenguaje, y asì lo reconoce intuitivamente todo programador. Los problemas de la integridad nacen con mayor frecuencia cuando el programador debe agregar funciones al programa, funciones que no pueden ser soportadas por el lenguaje que habìa creado anteriormente. Esto puede resultar exasperante, porque en muchos casos las funciones adiconales pueden implementarse facilemnte a través del mismo programa. El factor limitante siempre lo constituye la adición de nuevas expresiones en el lenguaje de entrada/salida.

Precisión

Todo lenguaje nuevo es difìcil de aprender. Ningún usuario tiene tiempo para perderlo en aprender un lenguaje innecesariamente florido. El lenguaje que un programador crea

para un programa debe ser directo y preciso al punto. Debe descansar lo más posible en las convenciones de comunicación, que ya conoce el usuario. Debe ser emocionalmente directo y obvio. Por ejemplo, la función CONTROL X es obscura. Qué significa? quizás se refiera a algo que debiera ser destruido; la X implica eliminación o negación. Tal vez implique que algo debiera ser examinado, expugnado, exhumado o algo similar. Si de hecho no se trata de ninguna de estas posibilidades, el comando es inaceptablemente indirecto. Los teclados se distinguen por crear este tipo de problemas.

Encierro

El encierro es uno de los aspectos del diseño de las comunicaciones, que produce los mayores problemas. El concepto se explica en la mejor forma a través de una analogìa. El usuario se encuentra en el punto "A" y desea usar el programa para llegar al punto "B". Un programa con una mala ingenierìa humana es como una cuerda floja tendida entre los puntos A y B. Si el usuario sabe exactamente lo que debe hacer y lo realiza a la perfección, tendrá éxito. Lo más probable sin embargo será, que resbale y caiga. Algunos programas tratan de ayudar, dando un manual o advertencias internas que indican al usuario, qué debe hacer y qué no debe hacer. Estos son análogos a los signos a lo largo de una cuerda floja que indiquen "tenga cuidado" o "no se caiga". He visto varios programas que colocan signos bajo la cuerda floja, de manera que el usuario al menos sabe por qué se está cayendo mientras lo hace. Programas de una clase un poco superior proveen máscaras contra ingresos ilegales, esto es equivalente a poner una baranda a lo largo de una cuerda floja. Son mucho mejores, pero deben estar muy bien construìdas para dar la seguridad que el usuario no vaya a botarlas. Algunos programas tienen mensajes odiosos que le ladran al usuario perdido, advirtiéndole de no hacer ciertos ingresos. Esto es lo mismo que los acechantes inspectores en los colegios de niños y solamente son útiles para hace sentir a un adulto como un niño. El programa ideal es como un túnel cavado a través de la roca sólida. No hay sino un camino, el camino que lleva al éxito. El usuario no tiene otra opción sino triunfar.

La esencia del encierro es el angostamiento de las opciones, la eliminación de posibilidades, la ubicación de paredes de roca sólida alrededor del usuario. El buen diseño no es un proceso acumulativo de ir apilando lotes de agregados sobre una arquitectura básica. El buen diseño requiere que el programador elimine las caracterìsticas menos importantes, opciones inútiles y trivialidades generales.

Esta tesis choca frontalmente con los valores de muchos programadores. Los programadores demandan libertad total para ejercer su poder sobre el computador. Su queja mas común contra un programa es que de alguna forma restringe sus opciones. Así, la deliberada recomendación del encierro se encuentra con incredulidad absoluta. Por qué debería ser alguien tan tonto como para restringir la potencia de esta herramienta tan poderosa?

La respuesta se encuentra en la diferencia entre el usuario y el programador. El programador dedica su vida al computador. El usuario es un conocido casual a lo más. El programador usa el computador en forma tan absoluta que le resulta económico tomarse el tiempo de aprender a usar una herramienta más poderosa. El usuario no tiene mucho tiempo para jugar con la máquina. Quiere llegar al punto B lo más rapidamente posible. No se preocupa de finezas del tipo que preocupan la vida del programador. Las bondades y ventajas aclamadas por los programadores sólo son una maraña para él. Ud. como programador puede no tener la misma escala de valores del usuario, pero si Ud. no quiere morirse de hambre, será mejor que se ciña a ella.

El encierro se logra creando entradas y salidas que no permitan valores prohibidos. Esto resulta extremadamente difícil de lograr con un teclado, ya que un teclado siempre tiene más entradas de las que un programa necesita. Este es un excelente argumento contra el uso del teclado. Un bastón es mucho mejor, porque es tan poco lo que se puede hacer con él y porque puede hacer tan poco es conceptualmente más fácil de excluir entradas malas. El ideal se consigue cuando todas las opciones necesarias se pueden expresar con el bastón y no quepan otras opciones. En este caso el usuario no puede producir una entrada equivocada, porque simplemente no existe. Aún más importante, como Newspeak (lingüaniva) en "1984" de Orwell, el usuario ni siquiera puede concebir pensamientos malos, porque no existen palabras (entradas) para ellos.

El encierro es mucho más que apantallar entradas malas. El apantallamiento permite que las entradas malas sigan siendo concebibles y expresables, aunque no funcionales. Por ejemplo, podría inhibirse la letra m de un teclado porque no tiene significado, aún así, el usuario podrá ver la tecla, podrá imaginar que la presiona y podría pensar que pasaría si llegara a presionarla. Todo un esfuerzo perdido. Incluso el usuario puede gastar más tiempo, presionando la tecla e imaginando por qué no sucedió nada. Esta pérdida de tiempo se ve complementada por el programador, imaginando al usuario hacer todas estas cosas y escribiendo los códigos para detener los síntomas, sin eliminar la enfermedad. Por contraste, una estructura de entrada apropiadamente encerrada hace uso de un

dispositivo de entrada, que puede expresar solamente las entradas necesarias para ejecutar el programa y nada más. El usuario no podrá así gastar tiempo imaginando algo que no existe.

Las ventajas que se logran cuando el encierro se aplica en forma apropiada son muchas. El programa resulta más compacto y se ejecuta con mayor rapidez, porque no hay necesidad de comprobación contra errores de ingreso, errores que están obsoletos en esta nueva sociedad, er, este programa. El usuario requiere menos tiempo para aprender el programa y tiene menos problemas al usarlo.

El principal problema con el encierro es el esfuerzo de diseño requerido para obtener un buen encierro. Toda la relación entre el programa y el usuario debe analizarse cuidadosamente, para determinar el mínimo vocabulario necesario, para que los dos se comuniquen. Habrá que examinar numerosos esquemas de comunicación y descartarlos antes de encontrar el verdadero esquema mínimo. En este proceso, muchos adornos, que el programador hubiese querido agregar, habrán de ser eliminados. Si el programador mira objetivamente más allá de su propia escala de valores, muchas veces concluirá que todos los adornos y agregados son más cosmética que esencia.

CONCLUSIONES

El diseño del lenguaje de comunicación entre el usuario y el programa es la parte mas difícil del proceso de diseño en los programas de usuario general. El diseñador debe contrapesar cuidadosamente las capacidades de la máquina y los requerimientos del usuario. Debe definir en forma clara la información que debe fluir entre los dos seres conscientes. Debe diseñar su lenguaje para maximixar la claridad (no la cantidad) de la información, que fluye hacia el usuario. Mientras que debe minimizar el esfuerzo que el usuario deba realizar para comunicarse con su computador. Su lenguaje debe aprovechar las posibilidades de la máquina en la forma más efectiva posible, manteniendo su propia integridad, su precisión y su encierro.

ALGUNOS PROBLEMAS COMUNES EN INGENIERIA HUMANA

Una vez discutido el problema de la ingeniería humana en términos teóricos, volvamonos ahora a discutir problemas de aplicación específicos en ingeniería humana. La lista de problemas no es exhautiva; simplemente cubre algunos de los problemas más comunes a la mayoría de los programas.

TIEMPOS DE RETARDO

Muchos programas requieren cálculos largos, de hecho la mayorìa de los programas ejecutan algún tipo de caáculo que requieren para su ejecución más de unos segundos. Qué experimenta el usuario mientras se ejecutan esos cálculos? Demasiados programas simplemente detienen el diálogo con el usuario por la duración del cálculo. El usuario queda abandonado frente a una pantalla inactiva y sin signo de vida por parte del computador. El computador no contesta a las entradas del usuario. Si el lenguaje de la comunicación entre el computador y el usuario es una creación de la ingenierìa humana, esta completa falta de comunicación sólo puede entenderse como una absoluta falta de ingenierìa humana. Dejar colgado asì al usuario, es absolutamente imperdonable.

Procesos separados

La mejor forma de entenderse con el problema de reconciliar los cálculos con la atención al usuario, es separar los procesos de entrada del proceso computacional. El usuario deberìa estar en condiciones de hacer sus ingresos mientras las computaciones avanzan. Esto puede lograrse tecnicamente, usando por ejemplo las interrupciones de borrado vertical (vea el apendice I). El programador puede realizar una multitarea con el procesamiento de entrada y el procesamiento principal. Esta técnica se emplea en FRENTE ORIENTAL 1941. El problema real de esta técnica es que muchos problemas son intrinsecamnte secuenciales en su naturalez. Es esencial para el usuario el entrar un valor o hacer una selección, antes de que los cálculos puedan avanzar hasta el paso siguiente. Esto hace difìcil de separar el procesamiento de las entradas del procesamiento principal. Sin embargo, es posible con un diseño astuto, hacer cálculos anticipados que determinarán valores intermedios, de modo que tan pronto se entren los datos crìticos, el resultado pueda obtenerse en forma mas rápida. Las aplicaciones de estas técnicas con seguridad pueden reducir los tiempos de retardo que experimenta el usuario.

Aceleración del programa

Otra forma de resolver el problema, es acelerar el programa mismo. Los códigos crìticos muchas veces pueden reescribirse para reducir los tiempos de ejecución. Un buen anidamiento de los bucles (el bucle de mayor cantidad de iteraciones deberìa ser el bucle interior a otro de menor cantidad de iteraciones) puede reducir el tiempo de ejecución. Una cuidadosa atención a los detalles de la ejecución puede apuntar a reducciones de tiempo adicional. Las mayores ganancias pueden lograrse convirtiendo BASIC en lenguaje assembler. El assembler es de 10 a 1000 veces más rápido que BASIC. La ventaja del Assembler se maximiza en las rutinas de movimiento de memoria y en la gráfico y algo menos para los

cálculos de coma flotante. Apantallando o enmascarando las interrupciones de borrado vertical, más tiempo de ejecución del 6502 puede liberarse para el procesamiento principal. Otras ganancias pueden lograrse reduciendo el tiempo fijo adicional, por acceso directo de memoria, que impone ANTIC. Esto puede lograrse, recurriendo a un modo gráfico simple (el Modo 3 de BASIC es el mejor); reduciendo la lista de despliegue también se puede reducir los costos del acceso directo a memoria. Apagando ANTIC totalmente es un camino muy drástico, que sólo crea problemas adicionales, al enfrentar al usuario con una pantalla en blanco.

Entretener al usuario

El tercer camino para manejarse con los tiempos de retardo, es mantener ocupado al usuario durante los cálculos. Un método de este tipo es una cuenta regresiva. El usuario ve la cuenta regresiva sobre la pantalla. Cuando ella llega a Ø, el programa vuelve para estar listo. Otro camino es trazar gráfica aleatoria sobre la pantalla. El perìodo de retardo siempre deberìa iniciarse con un cortés mensaje, avisando al usuario de este retardo. Deberìa terminar, por otro lado, con una campana u otro tipo de anuncio. No espere que el usuario mantenga sus ojos sobre la pantalla por un perìodo arbitrario de tiempo. El entretener al usuario durante el retardo, es una forma bastante pobre para manejar retardos que en principio no deberìan estar ahì, pero en todo caso es mejor que abandonar al usuario en su propia suerte.

LIDIANDO CON MALAS ENTRADAS DEL USUARIO

El problema más serio, que los programas para usuarios generales presentan, es la forma irresponsable en que manejan las entradas malas de usuario. El buen diseño elimina este problema, dando un lenguaje de entrada que no ofrezca ninguna posibilidad de entrada mala. Como ya lo indiqué antes, esto se logra en la mejor forma a través del uso de bastones. Sin embargo, hay aplicaciones (principalmente las de texto) que requieren un teclado, más aún, incluso los bastones ocasionalmente introducen problemas con las entradas de usuario. Cómo pueden manejarse estas entradas malas cuando no es posible purgarlas? Siguen algunas sugestiones. Es imperativo que cualquier esquema de protección se aplique consistentemente a través de todo el programa. Una vez que el usuario encuentre protección, la esperará en todos los casos. La falta de una protección de este tipo crea un abismo, al cual el usuario, pensando que esta seguro, caerá con toda probabilidad.

Indique el error y sugiera una solución

La forma más conveniente de enfrentar esta desagradable situación, es indicar el error del usuario en la pantalla en un lenguaje simple y sugerir una entrada correcta. Deben incluirse tres cosas en la respuesta del computador. Primero, la entrada del usuario debe reflejarse en la pantalla, de modo que él sepa lo que causó el problema. Segundo, la componente conflictiva de la entrada debe marcarse claramente y debe explicarse, de modo que el usuario sepa donde está lo malo. Tercero, debe sugerirse una alternativa de ingreso legal de modo que el usuario no se frustre por la sensación de que se encuentra frente a una pared de ladrillos. Por ejemplo, una respuesta apropiada para una presión de tecla equivocada podría ser así: "Ud. presionó CONTROL A, lo cual es solicitud de autopsia. No puedo realizar autopsias en gente viva. Sugiero que mate al sujeto primero".

Este método obviamente es muy caro en términos de tamaño de programa y tiempo de programación. Es éste el precio que uno paga por un mal diseño. Hay métodos menos costosos y menos efectivos.

Apantallando teclas prohibidas

Una solución común de los problemas de entrada por teclado es apantallar todas las entradas prohibidas. Si el usuario presiona una tecla equivicada, no sucede nada. No se genera el acostumbrado clic, ni aparece el carácter sobre la pantalla. El programa soóo escucha lo que quiere escuchar. Esta solución es segura en el sentido de que previene caìdas del programa, pero no protege al usuario de su confusión. Con seguridad el usuario sólo presionará una tecla si tiene la sensación de que algo va a suceder. Apantallar la tecla no corregirá la impresión equivocada del usuario. Sólo puede llevarlo a la conclusión de que algo realmente malo pasa con su computador. No queremos que esto suceda con nuestros usuarios.

Una variante a este esquema es agregar un zumbador odioso o una carraspera para castigar al usuario por su estupidez. De hecho, algunos programas de aficionados van tan lejos como para insultar por la vìa del texto al usuario. Estas técnicas son altamente cuestionables. Puede que existan casos que requieran ingresos de teclado peligrosos, que deban ser protegidos por mensajes fieros y odiosos, pero estos casos son muy pocos. Los mensajes correctivos deberìan siempre conformar a altos niveles de civilismo.

Mensajes de error

Una solución aún más barata es simplemente enviar un mesaje de error a la pantalla. Al usuario se le indica sólo que algo

hizo mal. En muchos casos el mensaje de error es crìptico y no ayuda para nada al usuario. El BASIC ATARI es un ejemplo extremo de esto. Los mensjes de error se dan por medio de n013meros solamente. Esto sólo se justifica cuando el programa debe operar bajo restricciones de memoria muy severas.

En la mayorìa de los casos el diseñador prefiere sacrificar caracterìsticas de ingenierìa humana, tal como un mensaje de error con sentido, a cambio de alguna potencia técnica adicional. Como se indicó al comienzo de este apéndice, estamos llegando a la etapa en que el poder técnico adicional ya no es un factor limitante al usuario, pero sì lo es la ingenierìa humana. Asì, el compromiso resulta menos justificable.

Compromisos entre protección y potencia

Una objeción a muchas de las caracterìsticas de ingenierìa humana, es que reducen la interacción entre el usuario y el computador. Los programadores se cansan de los incesantes "está Ud. seguro?" y restricciones similares. Una solución a este problema es la provisión de relaciones variables entre protección y potencia. Por ejemplo, un programa puede asumir un estado de alta protección durante la inicialización. Todas las entradas se comprueban cuidadosamente y se reflejan al usuario para su confirmación. El usuario tiene la opción de eliminar la protección y trabajar en un modo de alta velocidad. Esta opción no se hace obvia en la pantalla; se describe solamente en la documentación. Asì, el usuario intensivo puede trabajar al paso que le conviene y el usuario ocasional puede tener la protección adecuada.

LOS MENUS Y LAS TECNICAS DE SELECCION

Los menús son dispositivos normales para presentar al usuario las opciones disponibles. Son especialmente útiles para los usuarios novicios. Los esquemas orientados al comando, que son los preferidos por los programadores, suelan confundir al novicio que no puede darse el tiempo de aprender un diccionario de comandos usados por un programa orientado a los comandos. Existen varios problemas comunes asociados con el uso de los menús. Duscutiré algunos de ellos.

Tamaño del menú

Cuántas entradas debe haber en un menú? Obviamente el lìmite superior lo dicta el tamaño de la pantalla, pero este lìmite es demasiado grande, ya que en el modo BASIC 0 la pantalla puede tener hasta 48 entradas (24 lìneas con 2 alternativas por lìnea). Mi impresión es que 7 entradas es el lìmite superior deseable en el tamaño de un menú. Esto da lugar a bastante espacio en la pantalla para separar las entradas, dar un tìtulo del menú y agregar algún tipo de cnsulta.

Menús múltiples

Muchas veces un programa requerirá varios menús para cubrir totalmente las opciones que ofrece. Es muy importante que los menús múltiples puedan organizarse en una forma clara. El usuario facilmente puede perderse, vagando alrededor de estos mazos de menús. Una forma de lograrlo es tener un menú principal marcado claramente como tal y proveer cada menú secundario con una opción de regreso al menú principal. Otra forma es anidar los menús en una estructura jerárquica. Al recurrir a estos métodos, el programador debe proveer indicaciones de color y de sonido para ayudar al usuario a definir su posición dentro de la estructura de menús. Cada menú o cada nivel de menú debería tener una nota o un color distintivo asociado a él. La frecuencia de la nota debería estar asociada a la posición dentro de la jerarquía.

Métodos de selección

Una vez que el usuario ha visto sus opciones, cómo da a conocer su elección al computador? La forma más común es etiquetar cada entrada del menú con una letra o un número. El usuario realiza su selección presionando la tecla correspondiente en el teclado. Esta es una solución burda que comprende una indirección innecesaria. Hay una serie de métodos mejores. La mayoría de ellos usa el mismo esquema básico. Un puntero móvil apunta a una opción y un disparador lo selecciona. He visto un esquema que destaca la opción direccionada en video inverso. La tecla SELECT cambia el puntero, de manera que apunta a la siguiente selección del menú con rotación total desde el final del menú hacia el comienzo. La tecla START enclava una opción del menú. En otro programa he visto una rotación automática del puntero a través de las opciones del menú. El usuario solamente necesita presionar un botón en el instante correcto, cuando la opción deseada está siendo destacada (no me impresiono demasiado este método). Las paletas y los bastones se prestan muy bien para la selección de menús. Cualquiera de ellos puede usarse para barrer con el puntero a través de las selecciones del menú y el botón de disparo rojo para hacer la selección. Mi esquema preferido para la selección de un menú usa un cursor sobre un gran menú deslizante. El usuario mueve el cursor con el bastón. Hay señalizadores que pueden orientarlo hacia diferentes regiones del menú. El hace su selección, poniendo el cursor directamente sobre una opcion y presionando el botón disparador.

MANUALES Y TEXTOS DIRECTOS

Un problema común con los menús, mensajes de error, indicaciones y otros mensajes es que este material facilmente puede llegar a consumir una gran cantidad de memoria, memoria que mejor podría usarse para otras cosas. Este material podría colocarse en un documento de referencia, pero ello iría en

desmedro de la calidad de la ingeniería humana del programa. El diseñador deberá decidir cuanto material pondrá en el programa y cuanto de él debe ser relegado al manual. Con programas basados en discos es posible almacenar parte del material en el diskette. Esto disminuye la dureza del compromiso. Cuando el problema se mira solamente desde el punto de vista de la ingeniería humana, la respuesta es simple. Todo el material debe incluirse en el programa o al menos en un diskette. Las consideraciones ecónomicas y técnicas arguyen lo contrario. Es mi punto de vista personal, que cada tecnología debe usarse para las cosas que hace mejor. El computador, mientras es capaz de manejar el texto en forma estática, tiene su fuerte en el procesamiento de la información en forma dinámica. El papel y la tinta manejan la información estática, de modo más barato y muchas veces más claro que un computador. Por ello, prefiero poner la información estática en el manual y dejar que el programa refiera al usuario al manual. De todos modos incluyo la información crítica dentro del programa; mi línea divisoria se inclina según las necesidades del momento.

MEDIDAS DEL EXITO

Cómo puede un diseñador determinar el éxito de su ingeniería humana? Hay varios indicadores, que dan una realimentación valiosa. El primero es el largo mínimo del manual. Excluyendo el material general y aislando solamente el material del manual que es absolutamente necesario para describir el uso del programa, entonces el largo de este material es una buena medida de la ingeniería humana. Mientras más material, peor habrá sido el resultado. Un programa bien diseñado debe requerir muy poca explicación. Esto no debería usarse como argumento contra una buena documentación. La documentación siempre debe describir el programa en más detalle de lo absolutamente necesario. Un manual largo y acabado es bueno. Un programa que requiere un manual de este tipo, no lo es.

Otra medida es el tiempo que un usuario novicio gasta para aprender a usar el programa en forma satisfactoria. Los buenos programas pueden aprenderse en cosa de minutos.

Una tercera medida es la cantidad de reflexión que el usuario requiere para usar un programa. Un programa bien diseñado no debe requerir esfuerzo cognitivo para su uso. Esto no significa que el usuario no piensa para nada mientras usa el programa; más bien piensa acerca del contenido del programa y no en su mecánica. Debe concentrarse en qué está haciendo y no en cómo lo esta haciendo.

Un programa bien diseñado elimina la distancia mental entre el usuario y el computador. Los dos seres conscientes adquieren la sintonía mental, la comunión intelectual.

APENDICE III
LA GRABADORA DE PROGRAMAS EN CASSETTE ATARI

Es ésta una discusión de la Grabadora de Programas ATARI. Se cubrirán los siguientes puntos:

1. Cómo trabaja el cassette - información sobre los circuitos y los programas que se usan para operar el cassette.

2. Aplicaciones del cassette - como mezclar información de audio y digital para producir programas orientados hacia el usuario.

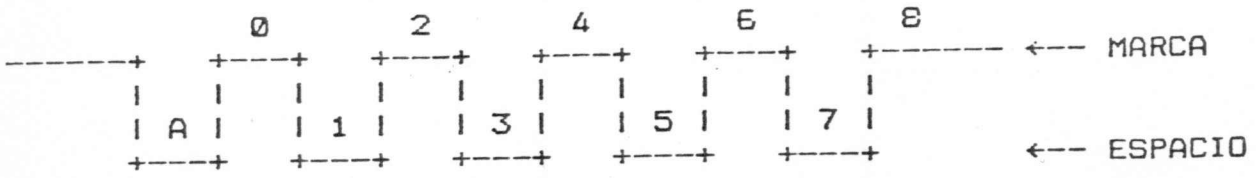
COMO TRABAJA EL CASSETTE

1.1 ESTRUCTURA DEL REGISTRO

Definición del byte:

El sistema operativo escribe archivos en bloques de largo fijo a razón de 600 baud (bits físicos/segundo). Se usa tranmisión serial asíncrona para leer y escribir datos entre el computador ATARI y la grabadora. POKEY reconoce cada byte de dato en este orden: 1 bit de partida (start) (espacio), ocho bits de datos (0=espacio, 1=marca), enseguida un bit de detención (stop) (marca). Los bytes se transmiten/reciben con su bit menos significativo primero.

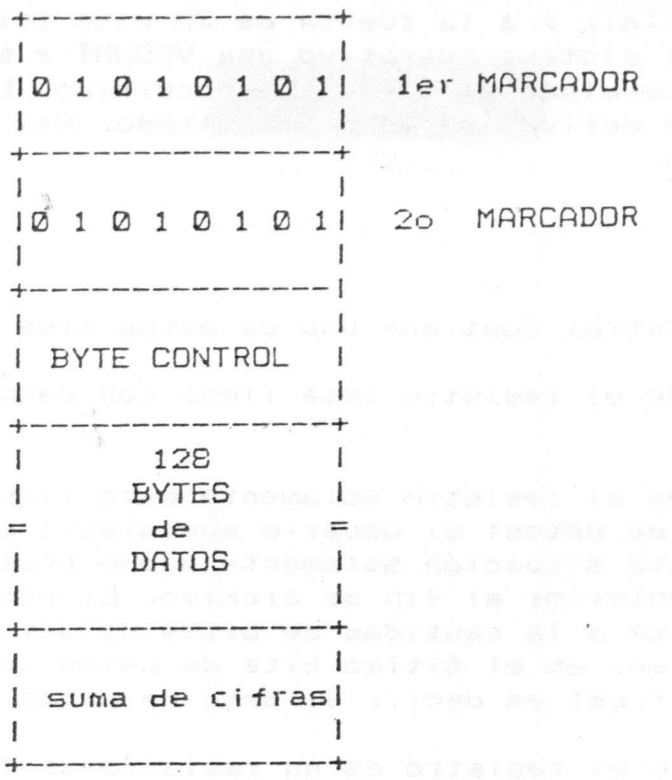
La frecuencia usada para representar una marca es de 5.327 Hz. Para el espacio la frecuencia corresponde a 3995 Hz. El formato de un byte de datos es como sigue:



A = Bit de partida (Start Bit), espacio
0-7 = Bits de datos
B = Bit de detención (Stop Bit), marca

Definición de registro:

Los registros tienen un largo de 132 bytes. Un registro se compone de los siguientes elementos: 2 caracteres de marca para medida de la velocidad, un byte de control, 128 bytes de datos y un byte de suma de cifras. El formato del registro se muestra a continuación:



1er y segundo MARCADORES

El carácter de cada marcador es un 55 (hexadecimal). Incluyendo los bits de partida y de detención, cada marcador tiene un largo de 10 bits. Desde el punto de vista ideal no debiera haber cinta en blanco entre los marcadores y los datos subsiguientes.

Medida de la Velocidad:

El propósito de los caracteres marcadores es ajustar la tasa de baud.

Se asume que la tasa de baud de entrada es nominalmente de 600 baud. Sin embargo, la rutina SIO efectúa un ajuste para compensar las variaciones de velocidad del motor, estiramientos de cinta, etc. Una vez calculados los valores verdaderos de la tasa de baud de recepción se efectúa el ajuste circuital. Teoricamente es posible mediante esta técnica, manejar tasas de baud de entrada en el rango de 318 a 1407 baud.

El sistema operativo verifica la velocidad de la cinta en la siguiente forma: observa continuamente la entrada de bit serial de POKEY, observando si aparece un bit de partida (bit 0) que indica el comienzo de un registro. Cuando encuentra uno, el sistema operativo guarda el valor del

contador de cuadro, almacenando el VCOUNT de ANTIC (contador vertical de pantalla). Continúa observando directamente la entrada de bit serial, y a la cuenta de 20 bits (final de los dos marcadores) el sistema operativo usa VCOUNT y el contador de cuadro para determinar el tiempo transcurrido. La tasa de baud necesaria se deriva de este resultado. Así se procede para cada registro.

Byte de Control:

El byte de control contiene uno de estos tres valores:

$FC indica que el registro está lleno con datos (128 bytes).

$FA indica que el registro solamente está lleno parcialmente de datos; el usuario suministró menos de 128 bytes. Esta situación solamente puede presentarse en el registro anterior al fin de archivo. El número correspondiente a la cantidad de bytes de datos, 1 a 127, se almacena en el último byte de datos anterior a la suma de cifras; es decir, el byte de datos # 128.

$FE indica que el registro es un registro de fin de archivo y está seguido de 128 bytes iguales a 0.

Suma de cifras:

La suma de cifras se genera y se verifica por medio de la rutina SIO, pero no está contenida en la memoria de transpaso entrada/salida del administrador de cassette, CASBUF [03FD].

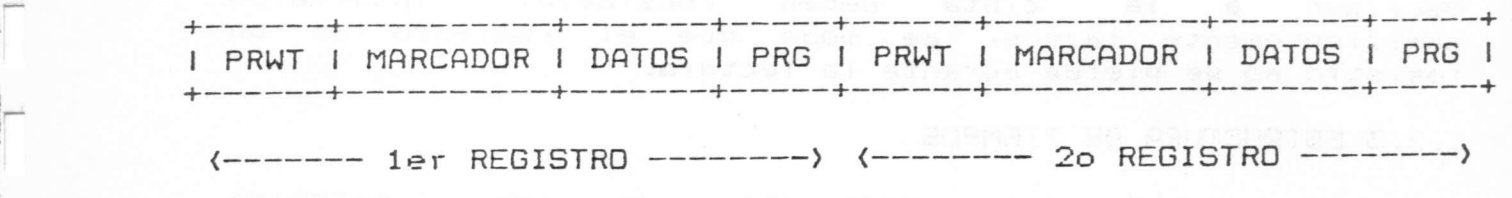
La suma de cifras es un byte suma de todos los demás bytes del registro, incluyendo los dos marcadores. La suma de cifras se calcula con transpaso de reserva (carry). A medida que cada byte se agrega a la suma, el bit de reserva también se agrega.

```
+-) suma parcial
| + byte de dato
| + bit de reserva
| ----------------
|
+-- resultado
```

1.2 SINCRONIZACION

1.2.1 INTERVALO ENTRE REGISTROS (INTER-RECORD GAP, IRG)

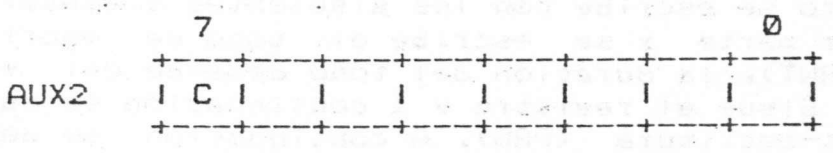
Como se mencionó en la sección 1.1, cada registro consta de 132 bytes de datos, incluyendo el byte de suma de cifras. Para distinguir un registro de otro, el administrador de cassette agrega un tono de escritura pre-registro (PRWT) y un intervalo post-registro (PRG). PRWT y PRG son ambos tonos de marca pura. Los intervalos inter-registro (IRG) entre cualquier par de registros consisten por tanto del PRG del primer registro, seguido del PRWT del segundo registro. Una representación gráfica de los registros y de intervalos es como sigue:



1.2.2 MODO IRG NORMAL Y MODO IRG CORTO

Los largos de PRWT y PRG dependen del modo de apertura de escritura. Hay dos tipos de IRG: modo IRG normal y modo IRG corto.

Al abrir un archivo, el bit más significativo de AUX2 especifica el modo. En la entrada o salida siguiente, el administrador de cassette ejecuta la escritura o lectura en el modo determinado por el MSB del byte AUX2:



C = 1 indica que el cassette debe ser escrito/leìdo en el modo de IRG corto.
C = 0 indica modo de IRG normal.

Modo IRG Normal:

Este modo se usa para las lecturas (READ) combinadas con procesamientos, es decir, la cinta siempre se detiene después de haberse leìdo un registro. Si el computador detiene la cinta y puede realizar su procesamiento con suficiente rapidez, la siguiente lectura puede ocurrir tan rapidamente, que la grabadora solamente capta una instantánea caìda de la lìnea de control.

Modo IRG Corto:

En este modo la cinta no se detiene entre registros, ya sea al escribir o durante la reproducción.

En reproducción el programa debe editar un READ para cada registro, antes de que pase frente a la cabeza de lectura. El único uso común, hasta ahora, de este modo es el almacenamiento de programas BASIC en forma interna (codificada), en la que durante la reproducción, BASIC no tiene nada que hacer con los datos, salvo ponerlos en RAM. Los comandos especiales de BASIC "CSAVE" y "CLOAD" especifican este modo.

Hay un problema potencial en esto. Los programas que escriben a la cinta deben considerar intervalos suficientemente largos, de modo que el comienzo de un registro no se pierda durante la lectura.

1.2.3 ESTRUCTURA DE TIEMPOS

Los tiempos para cada uno de los intervalos inter-registros son como sigue:

PRWT DEL INTERVALO INTER-REGISTRO NORMAL= 3 segundos de tono de marca.
PRWT DE INTERVALO INTER-REGISTRO CORTO= 0.25 segundos de tono de marca.

PRG DE INTERVALO INTER-REGISTRO NORMAL= hasta 1 segundo de tonos desconocidos.
PRG DE INTERVALO INTER-REGISTRO CORTO= Desde 0 hasta N segundos de tonos desconocidos, en que N depende de la sincronización del programa de usuario.

Cada registro se escribe con los siguientes tiempos: una vez que el motor parte y se escribe el tono de escritura pre-grabación (PRWT), la duración del tono depende del formato descrito arriba. Sigue el registro y a continuación se escribe el intervalo post-escritura (PRG). A continuación se detiene el motor para el modo normal, pero sigue escribiendo marca para el modo de intervalo inter-registros corto.

Note que para el modo de intervalo inter-registro normal, la cinta tendrá una sección de datos desconocidos debido a la detención y nueva partida del motor (es posible hasta 1 segundo de movimiento, dependiendo de la máquina usada). Estos datos desconocidos pueden ser tramos ilegibles de datos grabados previamente en la cinta.

1.2.2 CARACTERISTICAS DE ENTRADA/SALIDA CON SONIDO

La característica de entrada/salida con sonido es útil para determinar el éxito en la lectura de una cinta, particularmente con CLOAD. Las marcas y los espacios usan diferentes frecuencias de audio y uno pronto aprende a conocer los sonidos buenos y los sonidos malos que hace el sistema operativo.

1.3 LA ESTRUCTURA DE ARCHIVO

Un archivo consiste de los tres siguientes elementos:

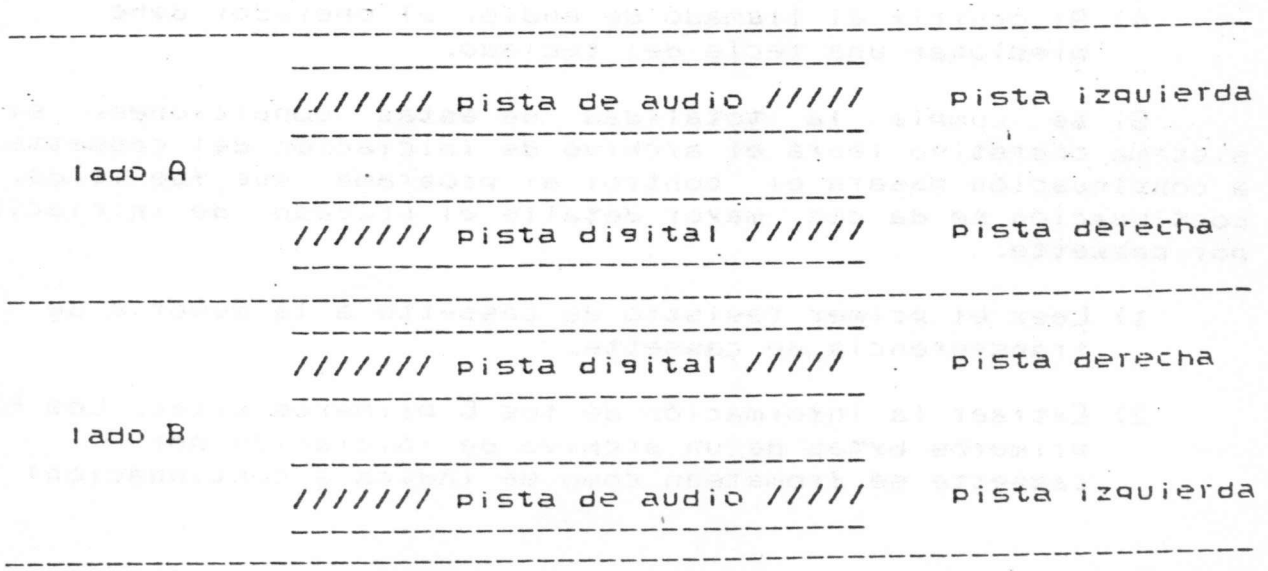
1) Una suìa de tono de marca de 20 segundos de duración.
2) Un número arbitrario de registros de datos.
3) un fin de archivo.

Cuando se abre el archivo (para salida) el sistema operativo parte por escribir una suìa de marca de 20 segundos, a continuación el sistema operativo vuelve al mandante, pero deja corriendo la cinta y escribiendo marcas.

El contador de tiempo lìmite de escritura/lectura está dispuesto para unos 35 segundos cuando el sistema operativo vuelve. Si el tiempo se excede antes de escribir el primer registro, la cinta se detendrá, dejando un intervalo entre la suìa abierta y la primera suìaa de registro.

1.4 ESTRUCTURA DE LA CINTA

Hay dos lados en cada cinta. Cada lado tiene 2 pistas, una para audio y otra para grabación digital. De esta forma la cinta puede grabarse en ambas direcciones. La siguiente es una vista en plano de la cinta:



Las cintas se graban en formato stereo de 1/4 de pista a 1 7/8 de pulgada por segundo (IPS). Recuerde que los ATARI usan grabadoras que tienen cabeza de configuración stereo (no simple o tipo monoaural).

1.5 LA INICIACION CON CASSETTE

El programa de autocarga en cassette puede inicializarse desde la grabadora, al energizarse el computador, como parte del sistema de inicialización.

La inicialización del sistema realiza funciones tales como poner en 0 todos los registros del circuito, limpiar la RAM, poner banderas, etc.

Después de haber ubicado todos los administradores residentes, si la tecla 'START' había sido presionada, se encuentra puesta la bandera CKEY [004A] de requerimiento de autocarga de cassette. Si esta bandera esta puesta, se intenta la inicialización de cassette.

Deben cumplirse los siguientes requerimientos para iniciar desde cassette:

1) El operador debe presionar la tecla 'START' mientras aplica potencia al sistema.

2) Debe encontrarse en la grabadora una cinta cassette con un archivo de autocarga del fromato apropiado, y la tecla 'PLAY' de la grabadora debe estar presionada.

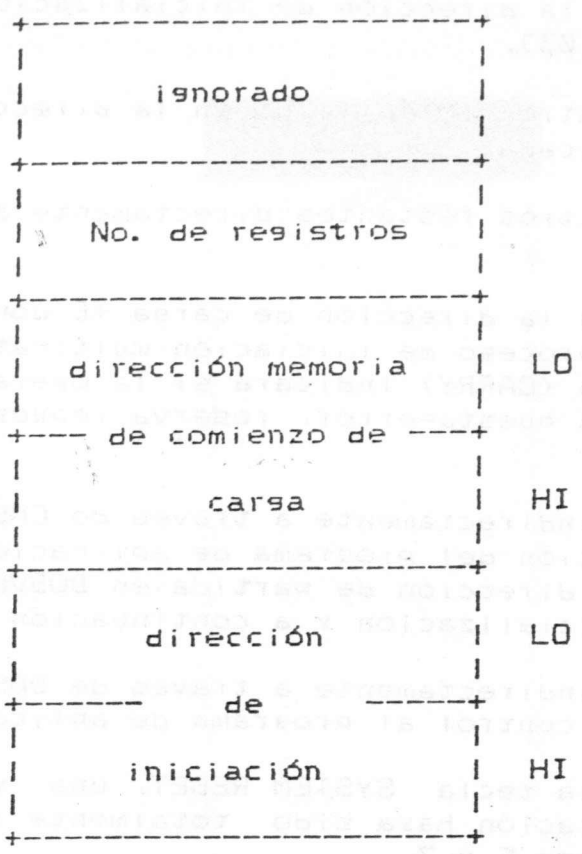
3) Dicho archivo de cassette debe haber sido creado en el modo de intervalo interregistro corto.

4) Al ocurrir el llamado de audio, el operador debe presionar una tecla del teclado.

Si se cumple la totalidad de estas condiciones, el sistema operativo leerá el archivo de iniciación del cassette y a continuación pasará el control al programa que fue leìdo. A continuación se da con mayor detalle el proceso de iniciación por cassette.

1) Leer el primer registro de cassette a la memoria de transferencia de cassette.

2) Extraer la información de los 6 primeros bytes. Los 6 primeros bytes de un archivo de iniciación por cassette se fromatean como se indica a continuación:



PRIMER BYTE: no se usa en el proceso de iniciación por cassette.

SEGUNDO BYTE: contiene el número de registros de cassette de 128 bytes cada uno, que deben leerse como parte del proceso de iniciación (incluye el registro que contiene esta información). Este número se encuentra en el rango de 1 a 255, en que 0 significa 256.

BYTES TERCERO Y CUARTO: contienen la dirección (LO,HI) a partir de la cual debe cargarse el primer byte del archivo.

BYTES QUINTO Y SEXTO: contienen la dirección (LO,HI) a la cual se transfiere el control, una vez completo el proceso de iniciación. El presionar la tecla SYSTEM RESET también transferirá el control a esta dirección, suponiendo que se ha completado el proceso de iniciación.

Una vez cumplido el paso 2, el programa de iniciación de cassette habrá:

A) anotado el número de registros de la iniciación.
B) almacenado la dirección de carga

C) almacenado la dirección de inicialización en CASINI [02,03].

3) Poner el registro recién leído en la dirección de carga especificada.

4) Leer los registros restantes directamente al área de carga.

5) Saltar (JSR) a la dirección de carga +6 donde podrá continuar un proceso de iniciación multi-etapa. El bit de reserva (CARRY) indicará si la operación tuvo éxito (reserva puesta=error, reserva repuesta=éxito) a su retorno.

6) Saltar (JSR) indirectamente a través de CASINI para la inicialización del programa de aplicación. Este debe poner su dirección de partida en DOSVEC [0A,0B] durante la inicialización y a continuación volver.

7) Saltar (JMP) indirectamente a través de DOSVEC para transferir el control al programa de aplicación.

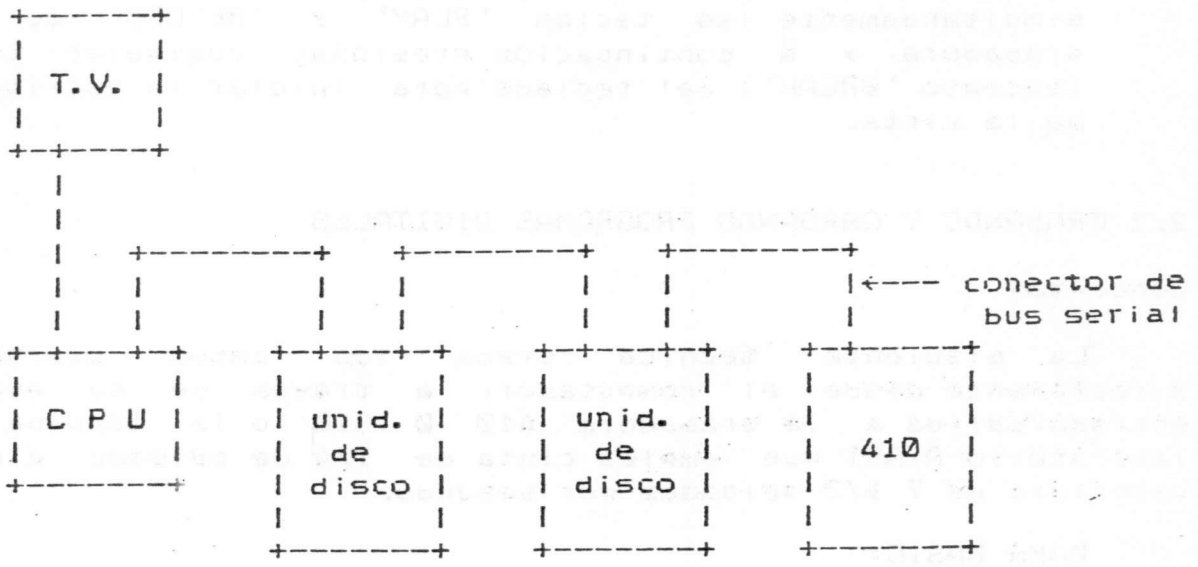
Si se presiona la tecla SYSTEM RESET, una vez que el programa de aplicación haya sido totalmente iniciado, se repetirán los pasos 6 y 7.

APLICACIONES DEL CASSETTE

Esta sección cubre la utilización del sistema de cassette Atari.

2.1 COMO CONFIGURAR EL SISTEMA DE CASSETTE.

La mayoría de los dispositivos de bus serial tienen dos conectores idénticos: uno corresponde a la entrada del bus serial y la otra al extensor de bus serial. Usando ambos conectores, los periféricos pueden concatenarse, simplemente cableándolos uno tras otro en forma secuencial, tal como en el siguiente diagrama:



Sin embargo, la grabadora cassette modelo ATP-410 no conforma a este protocolo de los demás periféricos, que recurren al bus serial. Este modelo de grabadora debe ser el último dispositivo del bus serial, porque no tiene conector de extensión como los demás periféricos y como también la grabadora ATP-1010. La falta de extensor de bus asegura que nunca pueda conectarse más de una grabadora cassette de este tipo en el sistema. Ello se consideró necesario, ya que el sistema no puede darse cuenta de la ausencia o presencia de una o más grabadoras cassette. Por otro lado, ello permite conectar, desconectar e intercambiar grabadoras cassette a voluntad. El modelo ATP-1010, si bien está provisto de conector de extensión de bus, desde el punto de vista del sistema operativo, tiene el mismo comportamiento del modelo ATP-410.

Cada vez que haya necesidad de abrir un archivo de cassette para lectura o escritura, el usuario debe seguir las siguientes instrucciones:

ENTRADA (DATOS DESDE 410 O 1010 AL COMPUTADOR): Cuando el cassette se abre para entrada, se genera un tono audible, perceptible en los modelos 400 y 800 a través del parlante interno. Si el cassette está dispuesto (energizado, cable de bus serial conectado, cinta dispuesta a comienzo del archivo), el usuario debe oprimir la tecla 'PLAY' de la grabadora y cualquier tecla (excepto la BREAK) en el teclado, para que se inicie la lectura de la cinta.

SALIDA (DATOS DESDE EL COMPUTADOR HACIA LA GRABADORA): Cuando la grabadora se ha abierto para salida, se generan dos tonos separados que, en el caso de los computadores 400 y 800, se escuchan a través del parlante interno. Si el cassette está dispuesto (como ya se describió más arriba), el usuario debe presionar simultaneamente las teclas 'PLAY' y 'RECORD' de la grabadora, y a continuación presionar cualquier tecla (excepto 'BREAK') del teclado para iniciar la escritura de la cinta.

2.2 GRABANDO Y CARGANDO PROGRAMAS DIGITALES

Concepto:

La siguiente técnica graba los datos digitales directamente desde el computador, a través de su puerta entrada/salida a la grabadora 410 ó 1010 o la máquina de laboratorio ATARI que emplea cinta de 1/4 de pulgada a una velocidad de 7 1/2 pulgadas por segundo.

PARA BASIC:

FORMATO: CSAVE
100 CSAVE

Este comando normalmente se emplea en el modo directo para grabar un programa residente en RAM a la cinta. 'CSAVE' escribe la versión codificada del programa en la grabadora Atari.

FORMATO: CLOAD
100 CLOAD

Este comando puede usarse, ya sea en modo directo o diferido para leer programas desde la cinta a la RAM para su ejecución.

PARA LENGUAJE ASSEMBLER:

PROGRAMA FUENTE
FORMATO: LIST#C:[,XX,YY]

Este comando se usa para escribir código fuente assembler. Los items en los paréntesis de opción [,XX,YY] indican la transferencia de solamente las líneas de XX hasta YY al cassette. Si no se dan los números de línea, se lista a cassette la totalidad del programa.

FORMATO: ENTER#C:

Este comando lee código fuente desde el cassette.

PROGRAMA OBJETO
FORMATO: SAVE#C:(XXXX,YYYY

Se graban en cassette los contenidos de un block de memoria entre ubicaciones XXXX e YYYY.

FORMATO: LOAD#C:

Este comando cargará en la memoria el material que se había grabado anteriormente. El rango de las ubicaciones de memoria que se llena será el mismo dado en el comando de grabación original.

2.3 GRABANDO PROGRAMAS DIGITALES CON AUDIO COMO FONDO

Concepto:

Esta técnica de grabación no permite ningún control de programa sobre el audio. El audio se hace sonar exclusivamente como música de fondo, para ayudar a pasar el tiempo durante el proceso de carga tan monótono.

PASO 1:
Siga las instrucciones de escritura digital dadas en 2.2 para los programas BASIC y Assembler, exceptuando que esta vez no se usa la cinta de cassette standard Atari (de 1 7/8 de pulgadas por segundo). Debido a que es difícil posteriormente para una persona grabar audio en una grabadora 410 o 1010, debemos usar la máquina de grabación de laboratorio Atari, que usa cinta maestra de 7 1/2 pulgada por segundo. Esta máquina de laboratorio es una máquina grabadora mucho más sofisticada, que permite grabar datos en una pista especifica.

Se conecta el modo de grabación para la pista derecha en la máquina de laboratorio, de modo que la información digital se grabe en la pista derecha de la cinta a 7 1/2 pulgada/seg.

PASO 2:

Repita el paso 1, ahora para la grabación de audio, excepto que primero debe rebobinar la cinta hasta el comienzo del programa y enseguida conmutar el modo de grabación para la pista izquierda. Así se graba el audio en la pista izquierda de la cinta a 7 1/2 pulgada/seg.

2.4 PROGRAMAS DIGITALES, AUDIO, MARCAS DE SINCRONISMO Y ADMINISTRACION DE PANTALLA

Concepto de las marcas de sincronismo:

No hay una forma eficaz para que un programa pueda detectar un segmento de audio mientras esté tocando el cassette. Para resolver este problema de sincronización se usa una marca de sincronismo, que informa al programa que determinado segmento de audio ya se ha ejecutado (un segmento de audio puede estar constituìdo, ya sea por una pieza musical o una instrucción, dependiendo de la aplicación).

En forma más precisa, como los datos de audio no tienen una estructura por registros, la marca de sincronismo grabada en la pista digital es más o menos lo mismo que una marca de fin de registro para el audio. Por ejemplo, una vez que el programa detecta la marca de sincronismo, puede decidir cual debe ser su siguiente acción, como detener el motor de la grabadora para dar tiempo para procesamiento o continuar y tocar el siguiente segmento de audio.

PASO 1:

El programador prepara un libreto de audio para "LA RANA". El libreto es asì:

(MUSICA) HOY DIA VOY A CONTARLES UNA HISTORIA QUE SE LLAMA "LA PRINCESA Y LA RANA". ES UNA HISTORIA MUY DULCE, DE MODO QUE NO SE VAYAN./

(MUSICA) ANTES DE COMENZAR CON MI HISTORIA, ME GUSTARIA SABER A QUIEN SE LA ESTOY CONTANDO. COMO TE LLAMAS? ESCRIBE TU NOMBRE Y PRESIONA RETURN. (PAUSA)

(MUSICA) AHORA COMENCEMOS CON LA HISTORIA. ERASE UNA VEZ UNA HERMOSA PRINCESA QUE VIVIA EN UN CASTILLO Y SU NOMBRE ERA YYYY./

(MUSICA) UN CLARO Y HERMOSO DIA, LA PRINCESA ESTABA PASEANDO POR ..../

NOTA:
"/" indica que el programa esta buscando una marca de sincronismo. Lo mejor es que el locutor se detenga alrededor de 1/2 segundo aquì, antes de continuar al siguiente segmento del libreto de audio.
"PAUSA" indica que el locutor se detiene aproximadamente 1 segundo para dar tiempo para detener y hacer partir otra vez el motor del cassette. Cada segmento de audio deberìa tener al menos 10 a 30 segundos de extensión, debido a que demasiadas marcas de sincronismo muy juntas pueden confundir al computador.

PASO 2:

Se sugiere que, antes de iniciar la codificación, el programador haga un plan general del programa, indicando las relaciones entre pantallas (CPU) y audio.

EJEMPLO: El siguiente ejemplo (vea la página III-16) ilustra como un programador puede crear un cassette con un programa que tiene control sobre una pista de audio. El ejemplo se llama "LA RANA":

PASO 3:

El programador comenzará codificando el programa llamado "LA RANA", que tendrá una apariencia como la siguiente:

```
 10 REM PROGRAMA "LA RANA" PARA DEMOSTRAR SINCRONISMO
 20 REM DE AUDIO CON PROGRAMA CON SISTEMAS DE CASSETTE
 30 REM
 40 DIM IN$(20)
 50 POKE 54018,52:REM PARTIDA MOTOR
 60 GRAPHICS 1
 70 PRINT #6;"LA PRINCESA Y LA RANA":PRINT #6....:REM
    PANTALLA PARA EVENTO 2
 80 GOSUB 1000:REM BUSQUEDA DE SINCRONISMO; VERIFICAR
    FIN INTRODUCCION HABLADA.
100 POSITION X,Y:PRINT #6;"TU NOMBRE ":REM PARA EVENTO 4
105 GOSUB 1000:REM EVENTO 5
110 POKE 54018,60:REM PARO MOTOR PARA INGRESO USUARIO
120 INPUT IN$:REM ESPERAR NOMBRE DEL USUARIO
130 POKE 54018,52
135 PRINT #6,CHR$(125):REM BORRAR PANTALLA
140 POSITION X,Y:PRINT #6;IN$:PRINT #6....:REM DESPLIEGUE
    PARA EVENTO 10
150 GOSUB 1000:REM VERIFICAR QUE LOCUCION PARA EVENTO 10
    HA TERMINADO
160 PRINT #6;....:REM LISTO PARA EVENTO 12
```

"LA RANA"

EVENTO	AUDIO	PANTALLA	VERIFIC. M.SINCR.	MODO MOTOR
1				SI
2	'HOY DIA..	LA PRINCESA Y LA RANA GRAFICA		
3			SI	
4	'ANTES DE..	LA PRINCESA Y LA RANA GRAFICA TU NOMBRE?XXX		
5			SI	
6				NO
7		ESPERAR HASTA RECONOCER UN INGRESO		
8				SI
9		BORRAR PANTALLA		
10	'AHORA....	XXX GRAFICA		
11			SI	
12	'UN CLARO..	GRAFICA		
13				

RUTINA DE BUSQUEDA DE MARCAS DE SINCRONISMO: En la cinta, la ausencia de sincronismo se representa por "MARCA" y la marca de sincronimso se representa por "ESPACIO". (Espacio es la frecuencia de un "0", que es de un tono más grave que el

sonido de marca que corresponde a una frecuencia "1". Como se mencionó antes, la frecuencia de marca es de 5327 Hz, y la de espacio es de 3995 Hz)] La rutina de búsqueda de marca de sincronismo permanentemente está pendiente de un "ESPACIO" en la puerta serial. La rutina se ve así:

```
1000 IF INT(PEEK(53775)/32+0.5)=INT PEEK(53775)/32) THEN
     RETURN:REM VERIFICAR EL 5o. BIT DE CADA BYTE
     ENTRANTE. SI ES UN "0", CORRESPONDE AL ESPACIO
     DE SINCRONISMO.
1010 GOTO 1000
```

RUTINA PARA CONTROLAR EL MOTOR: El programa puede echar a andar y detener el motor haciendo POKE en la ubicación 54018, con los datos dados a continuación:

```
MOVIMIENTO: POKE 54018,52
DETENCION : POKE 54018,60
```

PASO 4:

Una vez listo el esbozo del libreto de audio, el programador debe estimar los tiempos y largos de cintas requeridos para este libreto (incluyendo las pausas) y el programa. Si el largo de cinta requerido es demasiado para un cassette, entonces ya sea el libreto o el programa deben modificarse para hacerlos caber en un cassette.

PASO 5:

Grabe el programa a una cinta maestra, por ejemplo "MAESTRO 1".

PASO 6:

En base al libreto de audio se graba la voz con pausas en otra cinta maestra, "MAESTRO 2".

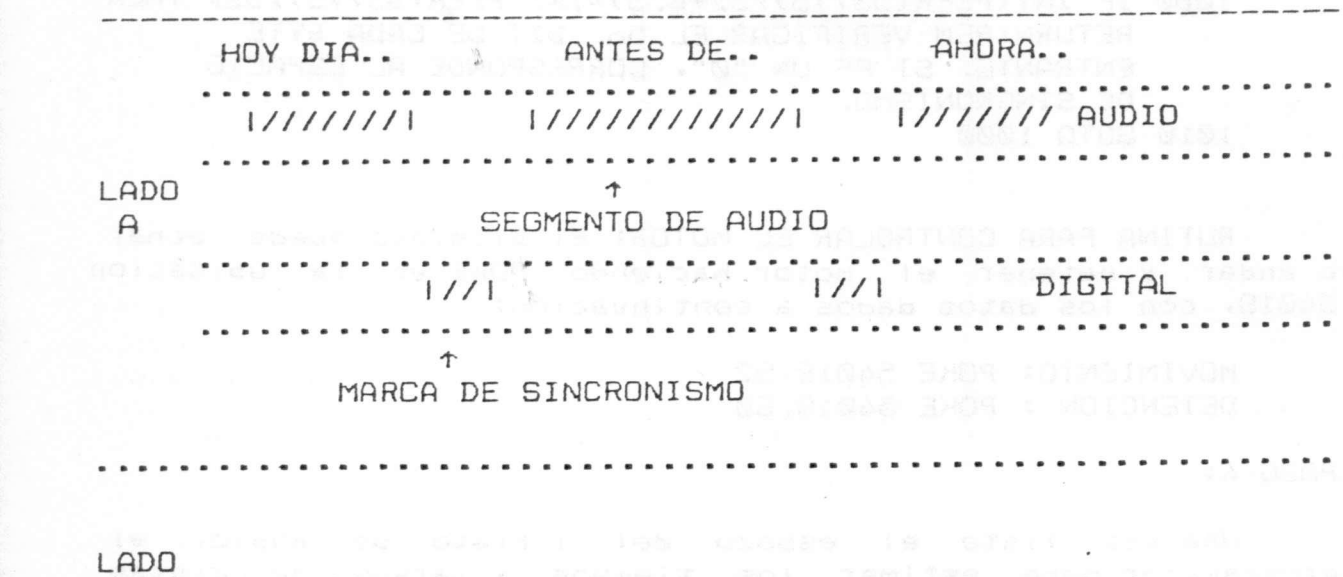
PASO 7:

Una vez producidos "MAESTRO 1" y "MAESTRO 2", ambas cintas se refunden para producir otra cinta maestra llamada "MAESTRO 3". "MAESTRO 3" tiene grabado en su comienzo el programa y a continuación el audio. Se necesitan tres máquinas de laboratorio para este procedimiento. Hasa 2 copias de "MAESTRO 3".

PASO 8:

Cargue el programa de marca de sincronismo en el computador. El propósito de este programa es escribir una marca de sincronismo continua (frecuencia "0") en la pista digital. La marca de sincronismo informa al programa que se

ha reproducido el segmento de audio. Cada vez que hay una pausa en el libreto de audio, se requiere una marca de sincronismo en ese lugar. La cinta terminada con audio y sincronismo será como sigue:



El programa de marcas de sincronismo se ve así

```
 10 REM PRESIONE LA TECLA (START) DEL COMPUTADOR
 20 REM PARA AGREGAR LA MARCA DE SINCRONISMO EN LA
    CINTA
 30 REM
 40 REM
 50 ES=53760:CONSOLA=53279:CASS=54018
100 FOR I=0 TO 8
110 READ J:POKE ES+I,J
120 NEXT I
125 REM LOS BUCLES DISPONEN FRECUENCIAS Y CANALES
    DE AUDIO
130 DATA 5,160,7,160,5,160,7,160,0
140 REM LA Entrada/Salida ESTA DISPUESTA
150 REM AHORA HAGA ANDAR EL CASSETTE
160 POKE CASS,52
200 POKE CONSOLA,8
210 IF PEEK(CONSOLA)<>7 THEN 230:REM CONSOLA=7
    SIGNIFICA ESCRIBIR "MARCA"
220 POKE ES+15,11:GOTO 200:REM NO HUBO PRESION DE
    (START)
230 POKE ES+15,128+11:GOTO 200:REM SI CONSOLA<>7
    ESCRIBIR "ESPACIO"
```

PASO 9:

Monte ambas cintas "MAESTRO 3" en dos máquinas grabadoras independientes y rebobine ambas máquinas a la unión entre programa y audio. Configure una grabadora con un computador que tenga cargado el programa de marcas de sincronismo. Esta grabadora está preparada para grabar la marca de sincronismo en la pista digital. La otra máquina grabadora reproducirá el audio grabado previamente.

PASO 10:

Digite "RUN" para iniciar el programa de marcas de sincronismo. Al mismo tiempo haga funcionar las máquinas grabadoras, una para grabar, la otra para reproducir. Escuche el audio y presione "START" cada vez que lo indique una pausa en el libreto de audio.

PASO 11:

Ahora la cinta está completa, con su programa grabado, seguido del audio y de las marcas de sincronismo. La cinta terminada está pronta para producción masiva.

2.5 INHIBIENDO LA TECLA BREAK

Se sugiere que el programador inhiba la tecla BREAK. Esto previene fallas del programa de cassette cuando el usuario accidentalmente presiona la tecla BREAK. El sistema operativo no es capaz de recuperar un registro parcial, a no ser que el usuario pueda rebobinar el registro perdido. La rutina de inhibición de la tecla BREAK es como sigue:

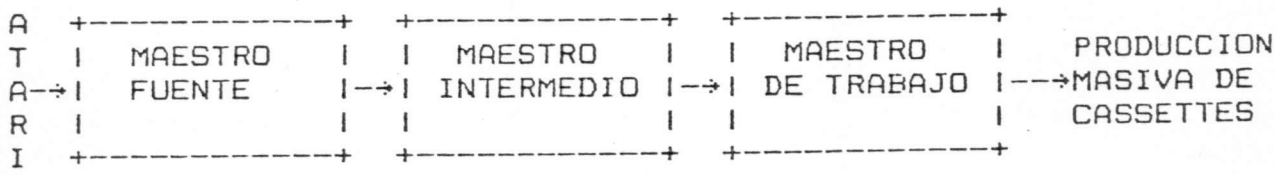
```
4000 X=PEEK(16):IF X<128 THEN 4020
4010 POKE 16,X-128:POKE 53774,X-128
4020 RETURN
```

La rutina de inhibición debiera ser llamada cada vez que haya un cambio de modo gráfico o un llamado de apertura de pantalla.

2.6 PRODUCCION MASIVA

El programador produce una o más cintas maestras, de acuerdo a las técnicas de grabación discutidas en las secciones 2.2, 2.3 y 2.4. Todas las cintas maestras Atari se graban en carrete abierto a 1/4 de pista, en cinta de 1/4 de pulgada grabada a 7 1/2 pulgada por segundo. La "CINTA MAESTRA" se suministra al duplicador como "MAESTRO FUENTE".

El duplicador tomará el "MAESTRO FUENTE" para hacer un "MAESTRO DE TRABAJO" para la producción masiva final de cassettes. El producto será de tercera generación con respecto al original. El siguiente es un diagrama de flujo del proceso:

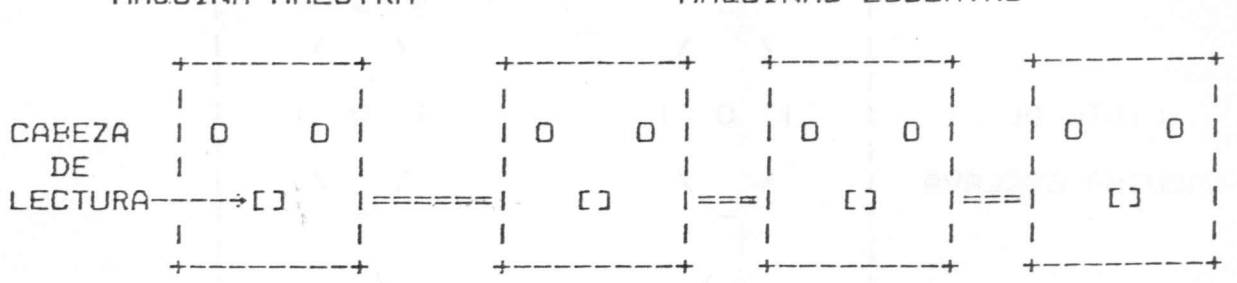


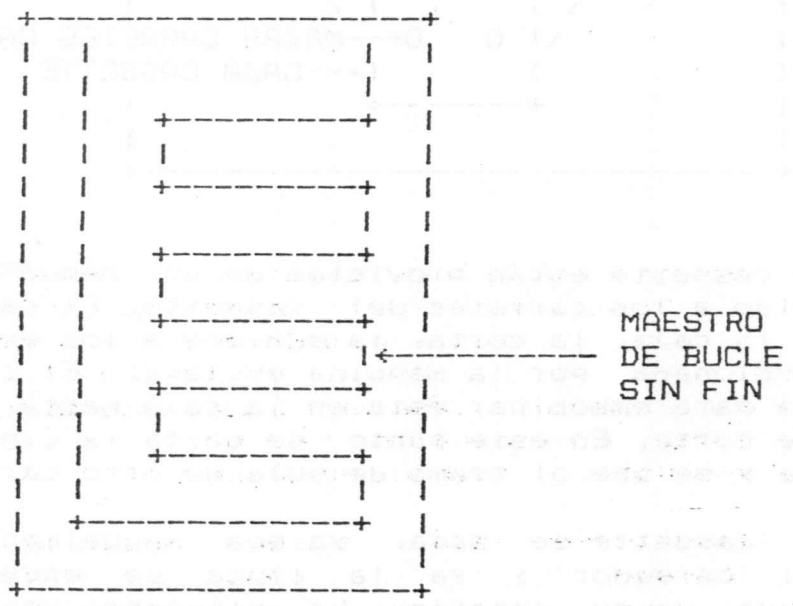
El "MAESTRO INTERMEDIO" se recomienda para el duplicador debido a que el "MAESTRO DE TRABAJO" puede destruirse o gastarse por uso excesivo. El "MAESTRO FUENTE" debería reservarse para usos solamente de emergencia. El "MAESTRO INTERMEDIO" es la copia de respaldo del "MAESTRO DE TRABAJO".

2.6.1 LA PRODUCCION MASIVA DE CASSETTES

Al presente, ATARI prefiere el método de bucle sin fin para producción masiva. El "MAESTRO DE TRABAJO" se copia para

producir un "MAESTRO DE BUCLE". El "MAESTRO DE BUCLE" puede ser de 1/4 de pulgada, 1/2 pulgada o de cualquier ancho de cinta. El bucle sin fin se une como bucle continuo con un corto trozo de cinta suìa en la unión. Se le coloca en una máquina maestra de bucle de alta velocidad que tiene una o varias máquinas esclavas. La configuración es asì:





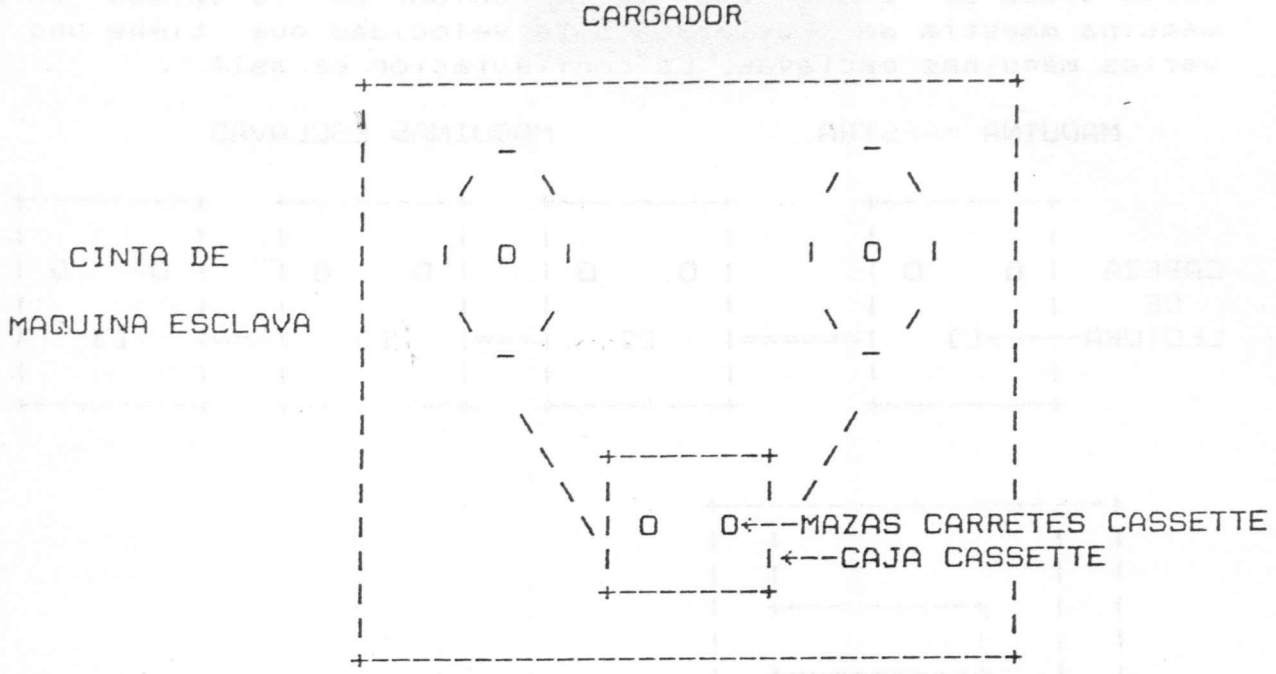
El "MAESTRO DE BUCLE" se lee repetitivamente. Si el duplicador desea producir 100 cassettes por ejemplo, el largo de la cinta en la máquina esclava se mide para igualar al largo del programa multiplicado por 100. Hay un contador en la máquina "MAESTRA" que se pone en 100.

A medida que el "MAESTRO DE BUCLE" se lee en forma continua, los datos (de las cuatro pistas) se copian a la cinta de la "MAQUINA ESCLAVA".

Al detectar la sección en blanco en el maestro de bucle, la máquina maestra genera un "TONO DE CORTE" que se graba en una o más pistas de las cintas de la máquina esclava. A continuación el contador aumentará en 1.

Cada cinta grabada en la máquina esclava tiene 100

programas grabados y 100 tonos de corte. La cinta se alimenta a una maquina cargadora automática que embobina las cintas en cajas de cassette C-Cero. La configuración es así:



Las cajas de cassette están provistas de un pequeño trozo de cinta guìa, unido a los carretes del cassette. El cargador tira esta guìa de la caja, la corta, la adhiere a los extremos de la cinta proporcionada por la máquina esclava. El carrete de la cinta se usa para embobinar ésta en la caja hasta que se detecta el tono de corte. En este punto se corta la cinta de la máquina esclava y se une al tramo de guìa de otro carrete.

La caja del cassette se saca, ya sea manualmente o mecanicamente del cargador y ya la cinta se encuentra totalmente enrollada en su interior. La siguiente caja de cassette se carga por el mismo proceso.

2.6.2 ENSAYO DE CONTROL DE CALIDAD

Cada vez que se inicia una producción deben tomarse muestras y verificarlas antes de dar la aprobación y ponerlas a la venta.

El ensayo de control de calidad normalmente se hace tomando el primer y el último cassette producido. Atari debe recibir al menos 10 muestras de cada producción masiva por cada maestro entregado.

APENDICE IV
ARTIFICIOS DE TELEVISION

Esta sección discute como obtener múltiples colores de un modo gráfico monocolor, a través del uso de los artificios de televisión.

Los modos ANTIC, mediante los cuales se puede lograr esto, son los 2, 3 y 15. El modo ANTIC 2 corresponde al BASIC 0, ANTIC 15 a BASIC 8, mientras el modo ANTIC 3 no tiene equivalente BASIC. Cada uno de estos modos tiene una resolución de pixel de 1/2 compás de color por una línea de barrido. Se consideran generalmente como de un color y dos luminancias. Recurriendo a los artificios, pueden desplegarse en la pantalla pixels de 4 colores diferentes en cada uno de estos modos.

El término de artificios de televisión se refiere a un punto o pixel en la pantalla que despliega un color diferente del asignado a él.

Un ejemplo simple de artificio, usando el computador ATARI, se muestra ingresando las siguientes líneas:

```
GRAPHICS 8
COLOR 1
POKE 710,0
PLOT 60,60
PLOT 63,60
```

Estas sentencias pintarán dos puntos sobre fondo negro; sin embargo, cada pixel tendrá un color diferente.

Para entender la causa de esta variación de color, primero debe entenderse toda la información de despliegue de TV que está contenida en una señal modulada de TV.

Las dos componentes principales de esta señal son la luminancia o brillantez y el color o tinte. La información de luminancia es la señal primaria y contiene no sólo los datos de brillantez, sino que también los de sincronismo y borrado horizontal y vertical. La señal de color contiene la información de color y se combina o modula sobre la forma de onda de la luminancia.

La luminancia de un pixel en la pantalla depende directamente de la amplitud de la señal de luminancia en ese punto. Mientras mayor la amplitud de la señal, más brillante sera el pixel.

La información de color, sin embargo, es una señal rotada en fase. Una señal rotada en fase es una forma de onda

permanentemente oscilatoria que ha sido retardada en alguna cantidad de tiempo en relación a una señal de referencia y este retardo de tiempo se traduce en color.

La señal de color oscila a una frecuencia constante de 3,579 MHz, definiendo en esta forma la máxima resolución de color horizontal de un televisor. Esto aparece en la pantalla en la forma de 160 ciclos de color, visibles a través de una línea de barrido (de hecho hay 228 ciclos de color, incluyendo los períodos de borrado y sincronismo horizontales y algún sobrebarrido).

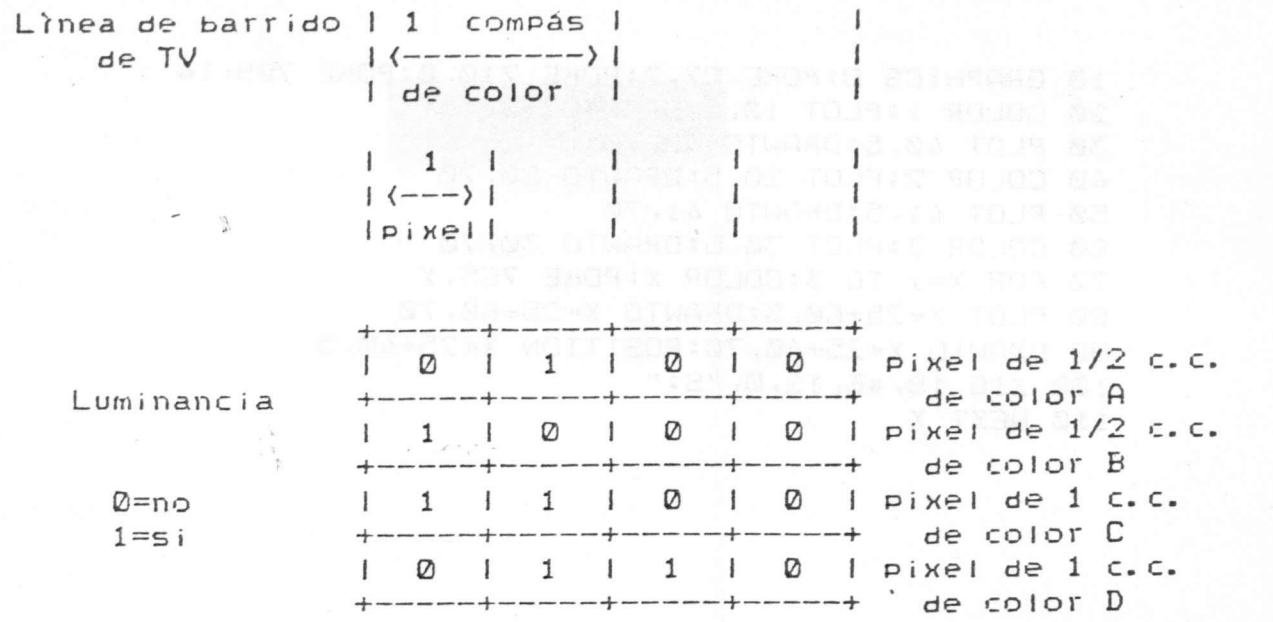
El término compás de color se refiere a un ciclo de color y es el término que se usa en general a través de la documentación ATARI, para describir unidades de medida a través de la pantalla. El modo gráfico 7 es un ejemplo de resolución de un compás de color, en que cada pixel de un compás de color puede tener un color diferente (hay sin embargo limitaciones del microprocesador).

ATARI también ofrece el modo de alta resolución (GRAPHICS 8) que despliega 320 pixels a lo largo de una línea. Esto se genera variando la amplitud de la señal de luminancia a razón de unos 7,16 MHz, que es el doble de la frecuencia de color.

Como ambas señales son teoricamente independientes, debería ser posible asignar un color de fondo en el despliegue y enseguida variar solamente la luminancia sobre la base de pixel por pixel. De hecho, este es el modo en que trabaja GR.8. El color de fondo procedente del registro de campo # 2 y las luminancias provenientes de los campos 1 y 2.

El problema está en que en la práctica las señales de color y luminancia no son independientes. Forman parte de una señal modulada que debe demodularse para usarse. Como la luminancia es la señal primaria, cada vez que cambia, tambien fuerza un cambio en la fase de color. Cuando hay uno o más compases de color de luminancia constante, esto no constituye problema, ya que la fase de color se mantendrá inalterable en esa área. Sin embargo, si la luminancia cambia en un límite de 1/2 compás de color, se producirá un desfase de color en ese punto. Más aún, este color no puede alterarse por parte del extremo transmisor de la señal (el computador ATARI).

Como la luminancia puede cambiar en límites de 1/2 compás de color, esto implicará que pueden generarse dos colores falsos o artificios del tipo pixel. Esto es basicamente cierto. Sin embargo, estos dos pixels pueden combinarse para formar dos tipos de pixels de compás de color completo. Ello se ilustra a continuación:



Note que cada uno de estos pixels requiere una distancia de un compás de color en el sentido horizontal, dando por tanto una resolución horizontal de 160.

Los colores de A a D dependen de cada televisor, generalmente porque la posición de la perilla de tinte cambia. Así, no pueden describirse como colores absolutos, por ejemplo rojo, pero definitivamente son diferentes uno del otro y se han escrito programas que utilizan estos colores.

Para ilustrar una simple aplicación de artificio, vea el ejemplo que sigue. Este programa traza líneas en cada uno de los cuatro colores artificiales y enseguida llena las áreas, usando tres de estos colores (note que el desplegar muchos pixels de ya sea el tipo C o el tipo D uno al lado del otro, resulta en lo mismo: una línea de luminancia constante de color fondo).

El POKE 87,7 hace que el sistema operativo considere a éste como un modo 7 y usa una máscara de 2 bits al disponer los bits en la memoria de despliegue. Para generar el color A, use COLOR 1, para color B use COLOR 2, y color C con COLOR 3. color D se genera al desplegar COLOR 1 a la izquierda de COLOR 2.

```
10 GRAPHICS 8:POKE 87,7:POKE 710,0:POKE 709,14
20 COLOR 1:PLOT 10,5:DRAWTO 10,70
30 PLOT 40,5:DRAWTO 40,70
40 COLOR 2:PLOT 20,5:DRAWTO 20,70
50 PLOT 41,5:DRAWTO 41,70
60 COLOR 3:PLOT 30,5:DRAWTO 30,70
70 FOR X=1 TO 3:COLOR X:POKE 765,X
80 PLOT X*25+60,5:DRAWTO X*25+60,70
90 DRAWTO X*25+40,70:POSITION X*25+40,5
100 XIO 18,#6,12,0,"S:"
110 NEXT X
```

APENDICE V
RUTINA EJEMPLO DE COMA FLOTANTE

```
0000                20       *=   $4000   ;PTO. DE PARTIDA ARBITRARIO
     =DDB6          30 FMOVE =    $DDB6
     =DA60          40 FSUB  =    $DA60
     =0482          50 FTEMP =    $0482
     =DDA7          60 FSTOR =    $DDA7
     =D8E6          70 FASC  =    $D8E6
     =00F3          80 INBUFF =   $F3
     =D800          85 AFP   =    $D800
     =00F2          90 CIX   =    $F2
     =0580          0100 LBUFF = $0580
                    0110 ;
     =009B          0120 CR  =    $9B
     =0009          0130 PUTREC = $09
     =0005          0140 GETREC = $05
     =E456          0150 CIOV =   $E456
     =0342          0160 ICCOM = $0342
     =0344          0170 ICBAL = $0344
     =0348          0180 ICBLL = $0348
                    0190 ; ORIGINAL ESCRITO POR CAROL SHAWW
                    0200 ; DEMO DE RUTINA DE COMA FLOTANTE (C.F.)
                    0210 ; LEE DOS NUMEROS DEL EDITOR
                           DE PANTALLA, LOS CONVIERTE A COMA
                           FLOTANTE,
                    0220 ; RESTA EL PRIMERO DEL SEGUNDO,
                           ALMACENA EL RESULTADO EN FTEMP
                           (REGISTRO C.F. DEL USUSRIO)
                    0230 ; Y DESPLIEGA EL RESULTADO
                    0240 ;
4000                0250 START
4000 205340         0260      JSR GETNUM  ; OBTENER PRIMER NUMERO
                                            DE E: Y CONVERTIR A C.F.
4003 20B6DD         0270      JSR FMOVE   ; MOVER NUMERO DE FR0 A FR1
4006 205340         0280      JSR GETNUM  ; OBTENER SEGUNDO NUMERO
                                            DE E: --OMITIR SI SOLO HAY
                                            UN ARGUMENTO
4009 2060DA         0290      JSR FSUB    ; FR0 (-- FR0 - FR1
                                            CAMBIAR SEGUN RUTINA DESEADA
400C 900A           0300      BCC NOERR   ; OMITIR SI NO HAY ERROR
                    0310 ;
                    0320 ; ERROR -- DESPLEGAR MENSAJE
                    0330 ;
400E A981           0340      LDA #MENSERR&255
4010 8D4403         0350      STA ICBAL
4013 A940           0360      LDA #MENSERR/256
4015 4C3940         0370      JMP CONTIN
4018                0380 NOERR
4018 A282           0390      LDX #FTEMP&255 ; ALMACENAR RESULTADO
                                                 EN FTEMP (VAR. C.F.
                                                 DE USUARIO)
401A A004           0400      LDY #FTEMP/256
401C 20A7DD         0410      JSR FSTOR
```

```
                        0420 ;
                        0430 ; CONVERTIR EL NUMERO A UN STRING ASCII
                        0440 ; ENCONTRAR FINAL DEL STRING, CAMBIAR
                               NO. NEGATIVO A POSITIVO Y AGREGAR (CR)
                        0450 ;
401F 20E6D8             0460     JSR FASC      ;CONVERTIR DE C.F. A
                                                STRING ASCII EN LBUFF
4022 A0FF               0470     LDY #$FF
4024                    0480 MLOOP
4024 C8                 0490     INY
4025 B1F3               0500     LDA (INBUFF),Y ;CARGAR BYTE
                                                 SIGUIENTE (APUNTADO POR
                                                 INBUFF). POSITIVO?
4027 10FB               0510     BPL MLOOP     ;SI. CONTINUAR
4029 297F               0520     AND #$7F      ;NO. NEGATIVO
                                                -- ELIMINAR BIT MAS SIG.
402B 91F3               0530     STA (INBUFF),Y
402D C8                 0540     INY
402E A99B               0550     LDA #CR       ;ALMACENAR (CR)
4030 91F3               0560     STA (INBUFF),Y
                        0570 ;
                        0580 ; DESPLEGAR RESULTADO
                        0590 ;
4032 A5F3               0600     LDA INBUFF    ;DIRECCION DE
                                                MEM. TRANS. ESTA EN INBUFF
4034 8D4403             0610     STA ICBAL
4037 A5F4               0620     LDA INBUFF+1
4039                    0630 CONTIN
4039 8D4503             0640     STA ICBAL+1
403C A909               0650     LDA #PUTREC   ;COMANDO PONER REGISTRO
403E 8D4203             0660     STA ICCOM
4041 A928               0670     LDA #40       ;LARGO MEM.TRANS = 40
4043 8D4803             0680     STA ICBLL
4046 A900               0690     LDA #0
4048 8D4903             0700     STA ICBLL+1
404B A200               0710     LDX #0        ;IOCB # = 0 (EDITOR
                                                DE PANTALLA
404D 2056E4             0720     JSR CIOV      ;LLAMAR CIO
4050 4C0040             0730     JMP START     ;LO MISMO DE NUEVO
```

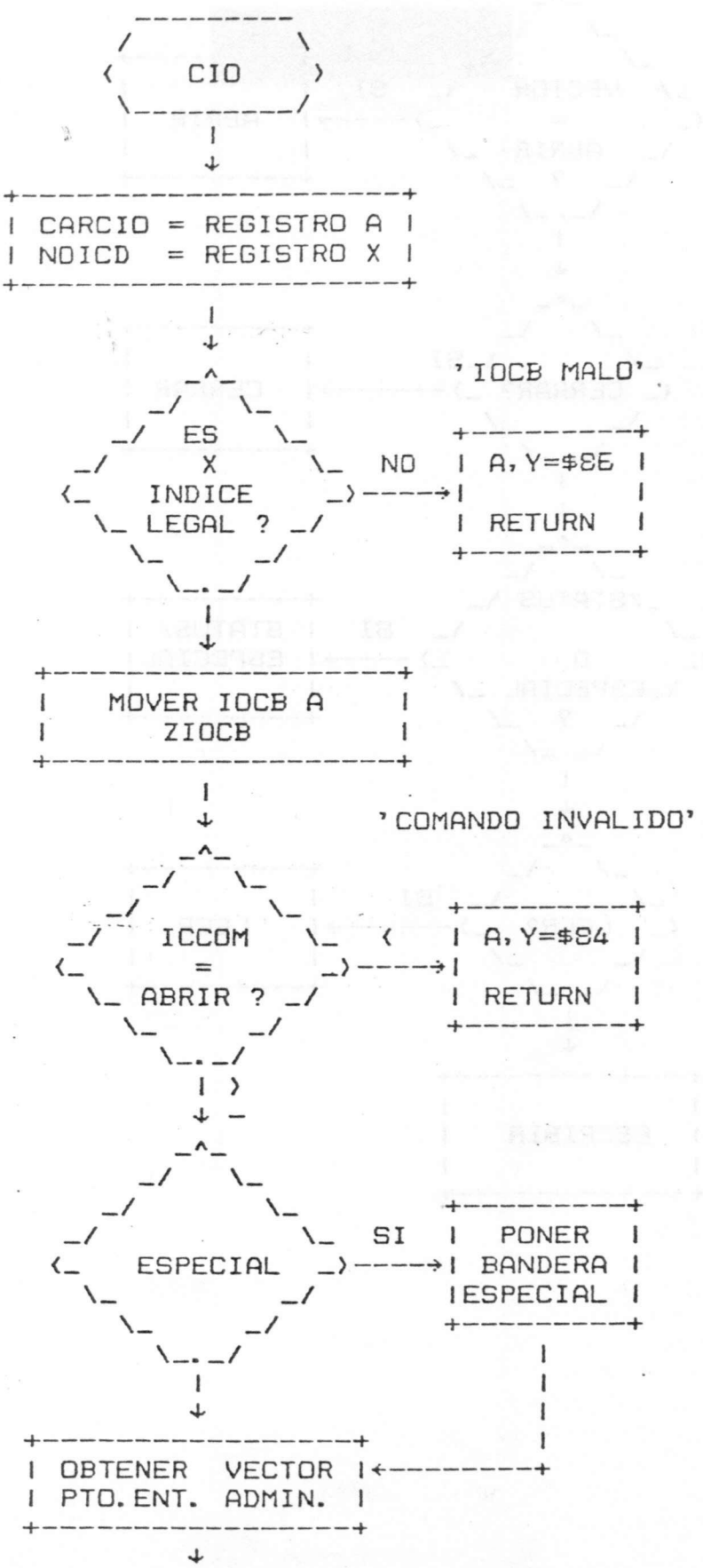
```
                    0740 ;
                    0750 ; GETNUM -- OBTENER STRING ASCII DE E:
                    0760 ;
4053                0770 GETNUM
4053 A905           0780     LDA #GETREC ;OBTENER REGISTRO
                                          (TERMINA EN(CR))
4055 8D4203         0790     STA ICCOM
4058 A980           0800     LDA #LBUFF&255 ;DIRECCION
                                            MEM.TRANS.=LBUFF
405A 8D4403         0810     STA ICBAL
405D A905           0820     LDA #LBUFF/256
405F 8D4503         0830     STA ICBAL+1
4062 A928           0840     LDA #40      ;LARGO MEM.TRANS. = 0
4064 8D4803         0850     STA ICBLL
4067 A900           0860     LDA #0
4069 8D4903         0870     STA ICBLL+1
406C A900           0880     LDA #0       ;ICBO # = 0
                                          (EDITOR DE PANTALLA)
406E 2056E4         0890     JSR CIOV     ;LLAMAR CIO
4071 A980           0900     LDA #LBUFF&255 ;ALMACENAR DIRECCION
                                            MEM.TRANS. EN
                                            PUNTERO (INBUFF)
4073 85F3           0910     STA INBUFF
4075 A905           0920     LDA #LBUFF/256
4077 85F4           0930     STA INBUFF+1
4079 A900           0940     LDA #0       ;INDICE MEM. TRANS. = 0
407B 85F2           0950     STA CIX
407D 4C00D8         0960     JMP AFP      ;LLAMAR ASCII A C.F.
                                           Y RETORNO
4080 60             0970 INIT RTS        ;RUTINA DE ENERGIZACION
                                          (NO HACER NADA)
4081 4552524F       0980 MENSERR .BYTE "ERROR",CR ;INDICA
                                                    CARRY PUESTO AL
                                                    RETORNO DE LA
                                                    RUTINA DE C.F.
4085 529B
                    0990 ;
                    1000 ; INFO DE PARTIDA DE LA RUTINA
                    1010 ;
4087                1020     *=  $02E0
02E0 0040           1030     .WORD START
02E0                1040     .END
```

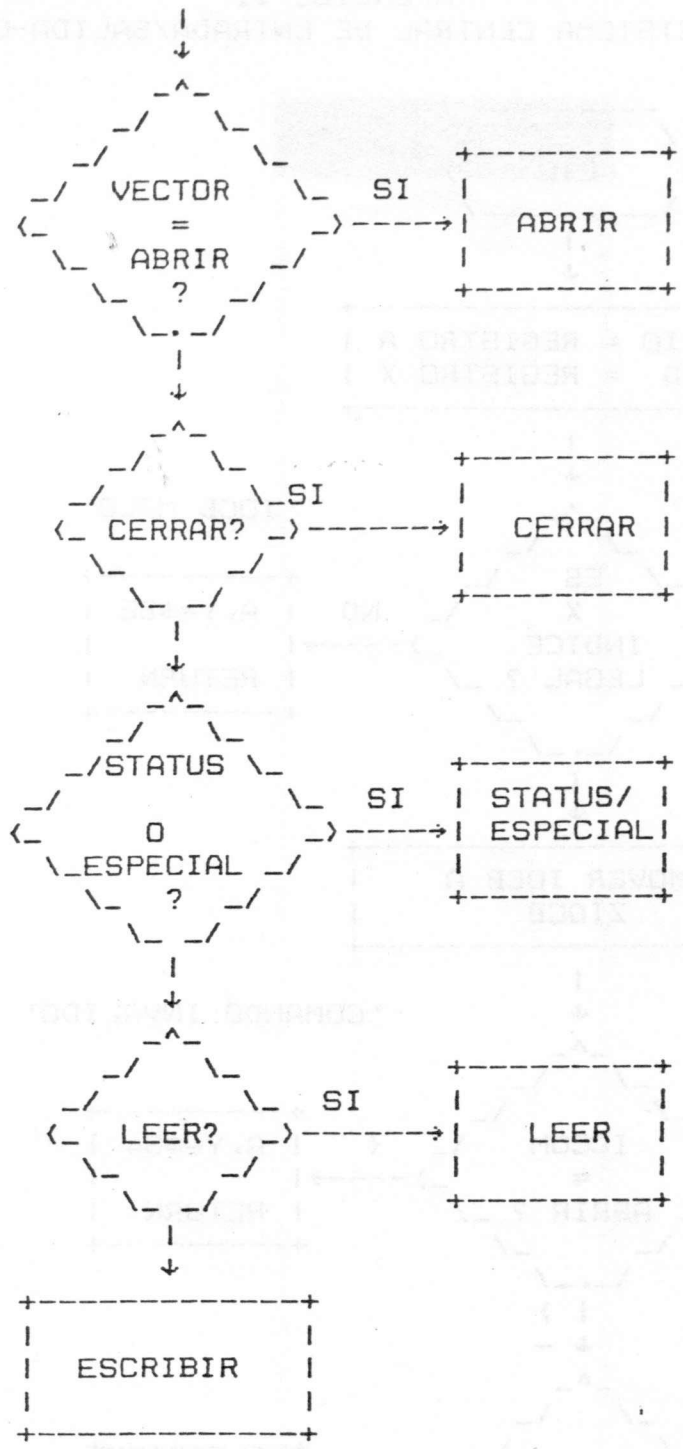
RUTINAS DE COMA FLOTANTE

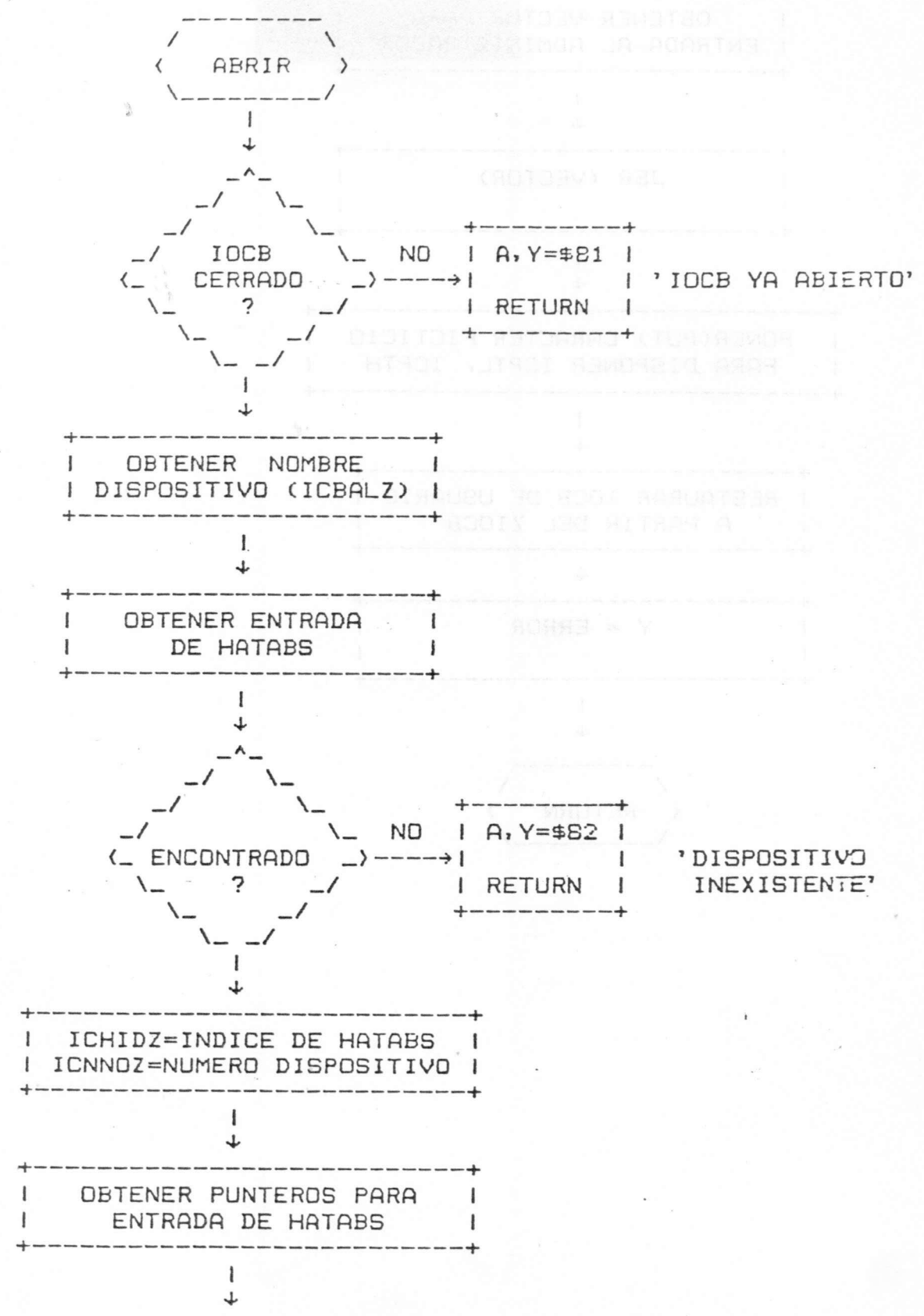
NOMBRE	DIREC.	DESCRIPCION	T.-MAX Apr. uSeg
AFP	D800	ASCII a coma flotante	3500
FASC	D8E6	Punto flotante a ASCII	950
IFP	D9AA	Integro a coma flotante	1330
FPI	D9D2	Punto flotante a integro	2400
FSUB	DA60	FR0 -- FR0 - FR1 Substracción	740
FADD	DA66	FR0 -- FR0 + FR1 Adición	710
FMUL	DADB	FR0 -- FR0 * FR1 Multiplicación	12000
FDIV	DB28	FR0 -- FR0 / FR1 División	10000
FLD0R	DD89	Carga C.F. de FR0 usando X,Y	70
FLD0P	DD8D	Carga C.F. de FR0 usando FLPTR	60
FLD1R	DD98	Carga C.F. de FR1 usando X,Y,	70
FLD1P	DD9C	Carga C.F. de FR1 usando FLPTR	60
FSTOR	DD7A	Almacenar F.C. FR0 usando X,Y	70
FSTOP	DDAB	Almacenar F.C. FR0 usando FLPTR	70
FMOVE	DDB6	FR0 -- FR1 Movimiento F.C.	60
PLYEVL	DD40	Evaluación de polinomio FR0	88300
EXP	DDC0	FR0 -- e exponenciación FR0	115900
EXP10	DDCC	FR0 -- 10 exponenciación	108800
LOG	DECD	FR0 -- LOG (FR0) log natural e	136000
LOG10	DED1	FR0 -- LOG (FR0) log común 10	125400
ZFR0	DA44	FR0 -- 0	80
AF1	DA46	Limp. reg.C.F. pag.0 (6 bytes)	80
en cartidge BASIC			
SIN	BDA7	FR0 -- sin(FR0)	79400
COS	BDB1	FR0 -- cos(FR0)	77400
ATAN	BE77	FR0 -- arctg(FR0)	126700
SQR	BEE5	FR0 -- raìz cuadrada de FR0	131100

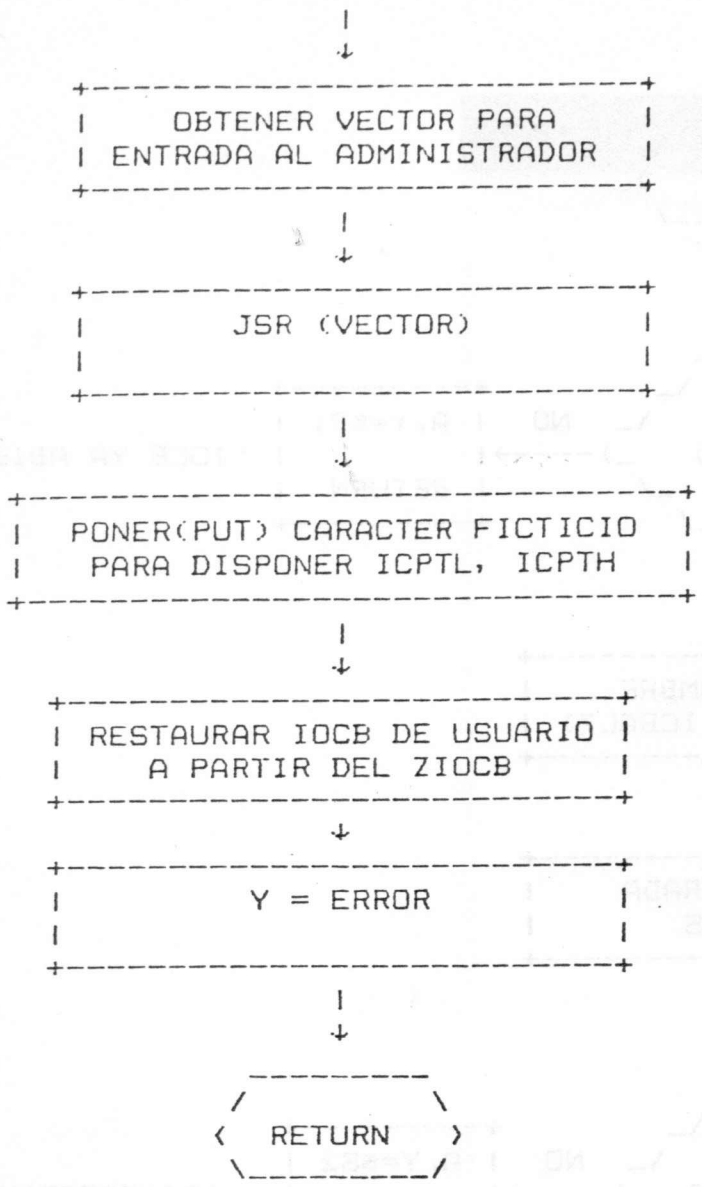
-Tiempos de 'caso más desfavorable', incluyendo JSR y RTS.
-Tiempos aproximados
-1 seg = 1.000.000 useg.
-Tiempos aproximadamente 30 a 40% menores al inhibir DMA (SDMCR): POKE 559,0 inhibe DMA
POKE 559,34 habilita DMA.
-FLPTR = Registro de Coma Flotante (Floating Point Register)

APENDICE VI
EL SISTEMA CENTRAL DE ENTRADA/SALIDA-CIO




VECTOR = ABRIR ?
SI
ABRIR
CERRAR? SI
CERRAR
STATUS 0 ESPECIAL
SI
STATUS/ ESPECIAL
LEER? SI
LEER
ESCRIBIR


ABRIR
IOCB CERRADO ?
NO
A,Y=$81
RETURN
'IOCB YA ABIERTO'
OBTENER NOMBRE DISPOSITIVO (ICBALZ)
OBTENER ENTRADA DE HATABS
ENCONTRADO ?
NO
A,Y=$82
RETURN
'DISPOSITIVO INEXISTENTE'
ICHIDZ=INDICE DE HATABS
ICNNOZ=NUMERO DISPOSITIVO
OBTENER PUNTEROS PARA ENTRADA DE HATABS


OBTENER VECTOR PARA
ENTRADA AL ADMINISTRADOR
JSR (VECTOR)
PONER(PUT) CARACTER FICTICIO
PARA DISPONER ICPTL, ICPTH
RESTAURAR IOCB DE USUARIO
A PARTIR DEL ZIOCB
Y = ERROR
RETURN

APENDICE VII
ADMINISTRADOR DE IMPRESOR QUME

```
01 ;EL ADMINISTRADOR USA LAS PUERTAS DE JUEGO PARA ENVIAR
   DATOS A UN IMPRESOR QUME
10 *=$3300
20 CR     =    $9B
30 SPCE   =    $20
40 PIAB   =    $D301
50 PIAC   =    $D303
60 DOSVEC =    $0A
70 DOSINI =    $0C
80 HATABS =    $031A
90 MEMLO  =    $02E7
0100 ;
0110 ;TABLA DE ENTRADA DEL ADMINISTRADOR
0120 ;
0130 QHTBL .WORD QOPEN-1
0140       .WORD QXIT-1 "CLOSE"
0150       .WORD QERR-1 "GET"
0160       .WORD QPUT-1
0170       .WORD QXIT-1 "STATUS
0180       .WORD QXIT-1 "SPECIAL"
0190       JMP QOPEN
0200 QHOOK1 LDA DOSINI    INICIAR AL CARGAR
0210       STA DOSLNK
0220       LDA DODINI+1
0230       STA DOSLNK+1
0240 QHOOK JSR QINST      TRATAR DE INSTALAR
0250 BCC   QHOO1          MODIFICANDO DOSINI
0260       RTS            PASO Y RETORNO
0270 QHOO1 LDA #QHOOK&255
0310       STA DOSINI
0320       LDA #QHOOK/256
0360       STA DOSINI+1
0370       RTS            VOLVER AL DOS
0380 QHOOK2 JSR JIND      PRIMERO INICIAR DOS
0390       JMP QHOOO      DESPUES INSTALAR ESTO
0400 JIND JMP (DOSLNK)    TAMBIEN INICIAR DOS
0410 DOSLNK .WORD $E477   ENLACE PARA INICIAR
0420 TONO .BYTE $80
0430 VTON .BYTE $60
0440 ACUH .WORD 0
0450 SPACES .WORD 0
0460 QWD .WORD 0
```

```
0470 ;
0480 ; INICIAR QUME Y
0490 ; VARIABLES DEL PROGRAMA
0500 ;
0510 QOPEN LDA #0
0520      LDX #5
0530 QILP STA ACUH,X      LIMPIAR VARIABLES
0540      DEX
0550      BPL QILP
0560 QSET LDA PIAC        DISPONER PUERTAS
0570      AND #$FB
0580      STA PIAC
0590      LDY #7
0600      STY PIAB        SACAR 3
0610      ORA #4
0620      STA PIAC
0630      LDA #2
0640      STA PIAB
0650      BNE QSNDU       RESTORE
0660 ;
0670 ; ENVIAR UNA PALABRA DE CONTROL AL QUME
0680 ;
0690 SNDYA STY QWD-1
0700 SNDA STA QWD
0710 QSND LDA PIAB
0720      AND #8
0730      BNE QSND        ESPERAR AVISO LISTO (RDY)
0740 QSNDU LDY #16
0750      LDA QWD
0760      EOR #7
0770      STA QWD
0780 QSLP LDA QWD
0790      AND #1
0800      ORA #2
0810      STA PIAB
0820      JSR QDELAY
0830      AND #1
0840      STA PIAB
0850      JSR QDELAY
0860      ORA #2
0870      STA PIAB
0880      LSR QWD+1
0890      ROR QWD
0900      DEY
0910      BNE QSLP
0920      JSR QDELAY
0930      ORA #4          STROBE
0940      STA PIAB
0950      JSR QDELAY
0960      AND #3
0970      STA PIAB
0980      RTS
```

```
0990 ;
1000 ; RETARDO ACOMODO DE LINEAS
1010 ;
1020 QDELAY LDX #60
1030 QDLPO DEX
1040      BNE QDLPO
1050      RTS
1060 ;
1070 ; ENVIAR UN CARACTER DESDE EL ACUMULADOR
1080 ;
1090 QPUT CMP #CR
1100      BEQ QCR
1110      CMP SPCE
1120      BCC RLY1
1130      BNE PUTPRT
1140      INC SPACES
1150 RLY1 BNE RLY2          (SIEMPRE)
1160 QCR LDA ACUH+1
1170      ORA #40           (NEGAR)
1180      TAY
1190      LDA ACUH
1200      ORA #2
1210      JSR SNDA          ENVIAR AL QUME
1220      LDA PASOV         (PASO VERTICAL)
1230      LDA #0
1240      JSR SNDYA         AVANCE DE LINEA (LF)
1250      LDA #0
1260      STA ACUH
1270      STA ACUH+1
1280      STA SPACES
1290 RLY2 JMP QXIT
1300 PUTPRT PHA             ALMACENAR CARACTER
1310      LDX SPACES
1320      BEQ PPX
1330      LDA #0
1340      TAY
1350 PPLP CLC
1360      ADC PASO
1370      BCC PPO
1380      INY
```

```
1390 PPO DEX
1400      BNE PPLP
1410      PHA
1420      STX SPACES
1430      CLC
1440      ADC ACUH
1450      STA ACUH
1460      TYA
1470      ADC ACUH+1
1480      STA ACUH+1
1490      PLA
1500      ORA #2
1510      JSR SNDYA
1520 PPX  LDA #0
1530      STA QWD+1
1540      PLA                 RECUPERAR CARACTER
1550      ASL A
1560      BCC NOUND
1570      LDY #5
1580      PHA
1590      LDA #$F1
1600      JSR SNDYA           SUBRAYAR AL REVES
1610      PLA
1620      CMP #SPCE+SPCE
1630      CLC
1640      BNE NOUND;          NO ES ESPACIO
1650      INC SPACES
1660      JMP QXIT
1670 NOUND ROL QWD+1
1680      ASL A
1690      ROL QWD+1
1700      ASL A
1710      ROL QWD+1
1720      ASL A
1730      ROL QWD+1
1740      ORA #1
1750      INC SPACES
1760      JSR SNDA
1770 QXIT  LDY  #1
1780      RTS
```

```
1790 ;
1800 ; SALIDA ERROR
1810 ;
1820 QERR LDY #$8B
1830      RTS
1840 ;
1850 ; INSTALAR ADMINISTRADOR EN HATABS
1860 ;
1870 QINST LDY #0
1880 QINLP LDA HATABS,Y
1890      CMP #$50          ENCONTRAR P:
1900      BEQ QIPUT
1910      INY
1920      INY
1930      INY
1940      CPY #$21          FIN DE LA TABLA?
1950      BCC QINLP         NO, BUCLE
1960      RTS               CASO CONTRARIO, RETORNO
1970 QIPUT LDA #QHTBL&255
2010      STA HATABS+1,Y
2020      LDA #QHTBL/256
2060      STA HATABS+2,Y
2070      LDA #QEND&255
2110      STA MEMLO         DISPONER MEMLOL
2120      LDA #QEND/256
2160      STA MEMLOH        DISPONER MEMLOH
2170      CLC
2180      RTS
2190 QEND =   *
2200      .END
```

APENDICE VIII
ACCESO ALEATORIO

DEFINICION DE ACCESO ALEATORIO

El acceso aleatorio se define como cualquier método para leer o escribir registros de/a cualquier parte de los datos de un archivo sin tener que leer previamente a través de registros previos de este archivo.

Antes de que se pueda procesar un comando de entrada/salida (I/O), el medio de almacenamiento debe ubicarse fisicamente en la posición del byte correcto con respecto al dispositivo. Un dispositivo secuencial, como lo es la grabadora de programas ATARI, debe ubicarse manualmente con ayda del contador de cinta por parte del operador. Así si se desea acceder al registro 100, la cinta debe pasar primero frente a 1 hasta 99. Resulta obvio que el procesamiento será muy lento, simplemente debido a las caracterìsticas fìsicas del dispositivo.

Un dispositivo aleatorio, como las unidades de disco ATARI, pueden ubicarse en cualquier byte de un archivo, bajo el comando de un programa BASIC. Así, si se desea el registro 100, se puede ubicar allì la unidad de disco de inmediato. En BASIC, el acceso aleatorio se logra por medio del comando POINT.

```
OPEN #1,12,0,"D:--------"
SECTOR=63
BYTE=26
POINT #1,SECTOR,BYTE
```

Los comandos anteriores hacen que el archivo abierto por el canal 1, se ubique en el sector 63, byte 26. Previamente debe abrirse el archivo en el modo 12 y el sector 63 debe haber sido asignado al archivo por el Sistema de Manejo de Archivos (FMS).

METAS

1. Definir una estructura de archivos de datos que facilite el acceso aleatorio.

2. Maximizar la utilización del diskette, usando todo el espacio disponible para el usuario en el archivo.

3. Simplificar la codificación, desarrollando subrutinas, que manejen el procesamiento de acceso aleatorio más frecuente.

CONCEPTOS

1. El acceso aleatorio puede facilitarse de varias maneras.

Primero, almacene la información de sector y byte, necesaria para el comando POINT, en el archivo de datos en forma tal que pueda ser asignada facilmente a una variable de programa al abrir el archivo. Segundo, haga que la variable del programa consuma sólo un mínimo de RAM.

Podría usarse un agrupamiento numérico de dos dimensiones como variable de programa. Sin embargo, cada registro requeriría 12 bytes de RAM. Una variable string cosume menos RAM, porque el número de sector, comprendido en el rango de 0 a 720, puede almacenarse en dos bytes y el desplazamiento de byte, comprendido entre 0 y 127, puede almacenarse en un byte.

2. Puede lograrse la máxima utilización del diskette, recurriendo al modo 12 (Entrada/Salida, I/O), en vez del modo 9 (agregar) en programas de mantención.

Hay dos problemas al usar el modo 9 para acceder un archivo de datos. Primero, al procesar el comando OPEN, el FMS siempre asignará un sector nuevo al archivo, independientemente de si el último sector se encuentra totalmente lleno de datos o no. Segundo, no funciona el comando POINT con archivos abiertos en el modo 9.

Se facilita el modo 12, al asignar registros en blanco al archivo de datos, al momento de crearlo.

3. La asignación de registros se facilita estableciendo un byte de status para cada registro. Un valor 0 indica que el registro no se encuentra activo; un valor 1 indica que el registro está activo.

ESTRUCTURA DEL ARCHIVO.

Un archivo de datos se compone de tres secciones.

REGISTRO DE ENCABEZAMIENTO DE ARCHIVO
PUNTEROS DE ACCESO DE ARCHIVO ALEATORIO
REGISTROS DE DATOS

El registro de encabezamiento de archivo es el primer registro del archivo. Tiene un largo de 125 bytes (124 datos + 1 limitador).

REGISTRO DE ENCABEZAMIENTO DE ARCHIVO

CONTENIDO DE BYTES
1-2 Dirección de sector del registro de encabezamiento de archivo
3 Desplazamiento de bytes del registro de encabezamiento de archivo
4 sin uso
5-6 # de registros del archivo
7-8 # de bytes de cada registro
9-124 sin uso

Los punteros de archivo de acceso aleatorio siguen inmediatamente después del registro de encabezamiento de archivo, de modo que comienzan en el segundo sector del archivo. Los datos se almacenan en forma de una variable string, cosistente de grupos de cuatro bytes. Los primeros cuatro bytes se usan para ubicar posición de los mismos strings de punteros en el archivo. A continuación vienen cuatro bytes para cada uno de los registros.

PUNTEROS DE ACCESO ALEATORIO DE ARCHIVO

CUENTA DE BYTES
1-2 dirección de sector de los punteros de acceso aleatorio de archivo
3 desplazamiento de bytes de los punteros de acceso aleatorio de archivo
4 sin uso
5-N 4 bytes para cada registro
1-2 dirección de sector del registro
3 desplazamiento de bytes del registro
4 status del registro

Los registros de datos siguen inmediatamente a continuación de los punteros de acceso aleatorio de archivo. Se almacenan como variables strings.

SUBRUTINAS DE ACCESO ALEATORIO

1. ABREARCH.SUB

Esta subrutina abre un archivo en el modo 12 e inicializa la svariables de acceso aleatorio que se usan en las demás rutinas.

Variables de entrada:

ARCH$ - debe dimensionarse y asignarse por parte del usuario.
CANAL - número del IOCB (1-5)

Llamado:

GOSUB 9300

Variables de salida:

MAXARCH - # de registros del archivo
LARARCH - # de bytes de cada registro
PUNARCH$ -contiene los punteros aleatorios
REGARCH$ -dimensionado para E/S de registros

Variables de trabajo:

SECARCH
BYTARCH no se usan en la actualidad
STSARCH

2. AGREARCH. SUB

Esta rutina asigna el siguiente registro disponible, leyendo a través de PUNARCH$, buscando un bit de status 0. Al encontrar un bit 0, se pone en 1 y el número del registro se almacena en REGISTRO. Si REGISTRO retorna un valor 0, significa que todos los registros del archivo están activos.

Variables de entrada:

ninguna

Lamado:

GOSUB 9400

Variables de salida:

REGISTRO -# del siguiente registro disponible
PUNARCH$ -puesto al dìa con byte de ststus 1

Variables de trabajo:

REGISTRO1
B

3. ELIARCH. SUB

Esta rutina señaliza los registros inactivos, cambiando a 0 su bit de status en PUNARCH$.

Variables de entrada:
REGISTRO

Llamado:
GOSUB 9450

Variables de salida:
PUNARCH$ - puesto al día con el bit de status 0

variables de trabajo:
B

4. PUNARCH.SUB

Esta subrutina actualiza la sección de los punteros de acceso aleatorio del archivo de datos, con el valor corriente de PUNARCH$.

Variables de entrada:
ninguna

Llamado:
GOSUB 9500

Variables de salida:
ninguna

Variables de trabajo:
S
B

5. UBIARCH.SUB

Esta rutina ubica el puntero del archivo al comienzo de un registro.

Variables de entrada:
REGISTRO -# número de registro al que debe apuntarse

Llamado:
GOSUB 9600

Variables de salida:
STS -valor del byte de status del registro

Variables de trabajo:
S
B

```
9200 REM APENDICE Y RUTINAS POR WILLIAM BARLETT
9300 REM 'ABREARCH.SUB' WBB 30-03-81
9305 REM ABRIR UN ARCHIVO EN MODO 12 Y DEFINIR
     TODAS LAS VARIABLES
9310 OPEN #CANAL,12,0,ARCH$
9315 DIM ENCARCH$(124)
9320 FOR I=1 TO 124:GET #CANAL,B:
     ENCARCH$(I)=CHR$(B):NEXT I:GET #CANAL,B
9325 SECARCH=ASC(ENCARCH$(1))*256+ASC(ENCARCH$(2))
9330 BYTARCH=ASC(ENCARCH$(3))
9335 STSARCH=ASC(ENCARCH$(4))
9340 MAXARCH=ASC(ENCARCH$(5))*256+ASC(ENCARCH$(6))
9345 LARARCH=ASC(ENCAERCH$(7))*256+ASC(ENCARCH$(8))
9350 DIM PUNARCH$(4+4*MAXARCH),REGARCH$(LARACH)
9355 FOR I=1 TO 4+4*MAXARCH:GET #CANAL,B:
     PUNARCH$(I)=CHR$(B):NEXT I:GET #CANAL,B
9360 RETURN


9400 REM 'AGRARCH.SUB' WBB 30-03-81
9405 REM ASIGNAR EL REGISTRO DISPONIBLE SIGUIENTE
9410 REGISTRO=0
9415 IF MAXARCH=0 THEN RETURN
9420 FOR REGISTRO1=1 TO MAXARCH
9425 B=REGISTRO*4+4
9430 IF PUNARCH$(B,B)=CHR$(0) THEN REGISTRO=REGISTRO1:
     REGISTRO1=MAXARCH:PUNARCH$(B,B)=CHR$(1)
9435 NEXT REGISTRO
9440 RETURN


9450 REM 'ELIARCH.SUB' WBB 30-03-81
9455 REM ELIMINA UN REGISTRO ACTIVO
9460 B=REGISTRO*4+4
9465 PUNARCH$(B,B)=CHR$(0)
9470 RETURN


9500 REM 'PUNARCH.SUB' WBB 30-03-81
9510 REM ESCRIBIR PUNARCH$
9515 S=ASC(PUNARCH$(1))*256+ASC(PUNARCH$(2))
9520 B=ASC(PUNARCH$(3))
9525 POINT #CANAL,S,B
9530 FOR I=1 TO 4+4*MAXARCH:B=ASC(PUNARCH$(I)):
     PUT #CANAL,B:NEXT I
9535 RETURN
```

```
9600 REM 'UBIARCH.SUB' WBB 30-03-81
9605 REM APUNTAR ARCHIVO AL REGISTRO
9610 S=ASC(PUNARCH$(REGISTRO*4+1))*256
     +ASC(PUNARCH$(REGISTRO*4+2))
9615 B=ASC(PUNARCH$(REGISTRO*4+3))
9620 STS=ASC(PUNARCH$(REGISTRO*4+4))
9625 POINT #CANAL,S,B
9630 RETURN
```

I. INTRODUCCION A ARCHOO1

Este archivo se emplea para crear un nuevo archivo de datos de disco, asignar su espacio de archivos e inicializar su estructura de acceso aleatorio. La estructura de acceso aleatorio está diseñada para su acceso desde las siguientes rutinas BASIC.

ABREARCH.SUB
AGRARCH.SUB
ELIARCH.SUB
PUNARCH.SUB
UBIARCH.SUB

II. ESTRUCTURA DEL PROGRAMA

0001-0999 lógica central
1000-9999 subrutinas

III. LOGICA DEL PROGRAMA. Las cinco fases de ejecución principales son: inicialización, definición del archivo, disposición de pantalla, asignación de archivos y cierre.

A. Inicialización 1000-1495

1015	dimensionamiento de variables
1020	identificación del programa al usuario
1025-1035	salida de usuario del programa
1040	disponer NADA$, para usaarse como relleno de strings

B. Definición del archivo 1500-1990

1510-1535	el usuario debe definir los parámetros
1540-1550	redefinición de parámetros para corregir
1600-1645	revisión del directorio del diskette, para verificar que el archivo especificado no existe todavìa
1700-1730	verificación de la disponibilidad de

suficientes sectores libres para la
creación del archivo
1800-1825 concatenación del nombre del archivo

C. Disposición de la pantalla 9600-9650

D. Asignación del archivo 2000-2499

2005 creación del archivo
2010 anotar punteros
2015 dimensionar variables
ENCARCH$ - registro encabezamiento archivo
PUNARCH$ - variable de puntero de archivo
REGARCH$ - variable de registro E/S
2100-2180 disponer ENCARCH$ y enviar al archivo
2200-2245 disponer PUNARCH$ y enviar al archivo
2350 actualizar pantalla
2355 enviar REGARCH$ al archivo
1370-2375 enviar último PUNARCH$ al archivo

E. CIERRE 0900-0999

```
10 REM 'ARCH0001' WBB 30-03-81
100 REM LINEA PRINCIPAL
110 GOSUB 1000
120 IF SN$="N" THEN 900
130 GOSUB 1500
140 GOSUB 9600
150 GOSUB 2000
900 REM FIN
910 CLOSE #1
920 GRAPHICS 0
930 END
1000 REM INICIO
1005 TRAP 9800
1010 GRAPHICS 2
1015 DIM SN$(1),DIS$(15),ARCH$(8),EXT$(3),NOMARCH$(15),
     FMS$(20),NADA$(128)
1020 PLOT 5,4:PRINT #6;"ARCH0001"
1025 PRINT "ESTE PROGRAMA INICIA UN NUEVO ARCHIVO."
1030 PRINT "QUIERE SEGUIR ADELANTE (S/N) ";
1035 INPUT SN$
1040 NADA$=" ":NADA$(128)=" ":NADA$(2)=NADA$
1095 RETURN
1500 REM DEFINIR ARCHIVO
1505 GRAPHICS 2
1510 PRINT #6;"DEFINIC. DEL ARCHIVO"
1515 PRINT "    DISPOSITIVO ";:INPUT DIS$:
     DIS$(LEN(DIS$)+1)=":"
1520 PRINT " NOMBRE ARCH. ";:INPUT ARCH$
1525 PRINT "    EXTENSION ";:INPUT EXT$
1530 PRINT "    REGISTROS ";:INPUT MAXARCH
1535 PRINT "   LARGO REG. ";:INPUT LARARCH
1540 PRINT "QUIERE CONTINUAR (S/N) ";
1545 INPUT SN$
1550 IF SN$<>"S" THEN 1500
1600 REM VERIFICAR QUE EL ARCHIVO NO EXISTE
1605 FMS$=DIS$:FMS$(LEN(FMS$)+1)="*.*"
1610 OPEN #1,6,0,FMS$
1615 INPUT #1,FMS$
1620 IF FMS$(2,2)<>" " THEN 1700
1625 IF FMS$(3,2+LEN(ARCH$))<>ARCH$
     OR FMS$(11,10+LEN(EXT$))<>EXT$ THEN 1615
1630 CLOSE #1
1635 PRINT "ESTE ARCHIVO YA EXISTE!";CHR$(253)
1640 GOSUB 9850
1645 GOTO 1500
1700 REM VERIFICAR SI EL DISCO TIENE SUFICIENTE ESPACIO
1705 CLOSE #1
1710 LIBRE=VAL(FMS$(1,3)):USO=MAXARCH*(LARARCH+5)/125+1
1715 IF USO<LIBRE THEN 1800
1720 PRINT "INSUFICIENTE ESPACIO DE DISCO":
     PRINT LIBRE;" LIBRE, SE NECESITA ";USO;CHR$(253)
1725 GOSUB 9850
```

```
1730 GOTO 1500
1800 REM CONCATENAR NOMBRE ARCHIVO
1805 NOMARCH$=DIS$
1810 NOMARCH$(LEN(NOMARCH$)+1)=ARCH$
1815 NOMARCH$(LEN(NOMARCH$)+1)="."
1820 NOMARCH$(LEN(NOMARCH$)+1)=EXT$
1825 RETURN
2000 REM INICIAR ENCABEZAMIENTO,PUNTERO,STRINGS REGISTRO
2005 OPEN #1,8,0,NOMARCH$
2010 NOTE #1,SECARCH,BYTARCH
2015 DIM ENCARCH$(124),PUNARCH$(4+4*MAXARCH),
     REGARCH$(LARARCH)
2100 REM ENCABEZAMIENTO ARCHIVO
2105 ENCARCH$=NADA$
2110 HI=INT(SECARCH/256)
2115 LO=SECARCH-HI*256
2120 ENCARCH$(1,1)=CHR$(HI)
2125 ENCARCH$(2,2)=CHR$(LO)
2130 ENCARCH$(3,3)=CHR$(BYTARCH)
2135 ENCARCH$(4,4)=CHR$(0)
2140 HI=INT(MAXARCH/256)
2145 LO=MAXARCH-HI*256
2150 ENCARCH$(5,5)=CHR$(HI)
2155 ENCARCH$(6,6)=CHR$(LO)
2160 HI=INT(LARARCH/256)
2165 LO=LARARCH-HI*256
2170 ENCARCH$(7,7)=CHR$(HI)
2175 ENCARCH$(8,8)=CHR$(LO)
2180 PRINT #1;ENCARCH$
2200 REM PUNTERO ARCHIVO
2205 FOR I=1 TO 4+4*MAXARCH STEP 128:PUNARCH$(I,I)=NADA$:
     NEXT I
2210 NOTE #1,S,B
2215 HI=INT(S/256)
2220 LO=S-HI*256
2225 PUNARCH$(1,1)=CHR$(HI)
2230 PUNARCH$(2,2)=CHR$(LO)
2235 PUNARCH$(3,3)=CHR$(B)
2240 PUNARCH$(4,4)=CHR$(0)
2245 PRINT #1;PUNARCH$
2300 REM REGISTROS
2305 FOR I=1 TO LARARCH STEP 128:REGARCH$(1)=NADA$:NEXT I
2310 FOR I=1 TO MAXARCH
2315 NOTE #1,S,B
2320 HI=INT(S/256)
2325 LO=S-HI*256
2330 PUNARCH$(I*4+1,I*4+1)=CHR$(HI)
2335 PUNARCH$(I*4+2,I*4+2)=CHR$(LO)
2340 PUNARCH$(I*4+3,I*4+3)=CHR$(B)
2345 PUNARCH$(I*4+4,I*4+4)=CHR$(0)
2350 GOSUB 9700
2355 PRINT #1;REGARCH$
2360 NEXT I
2365 CLOSE #1
```

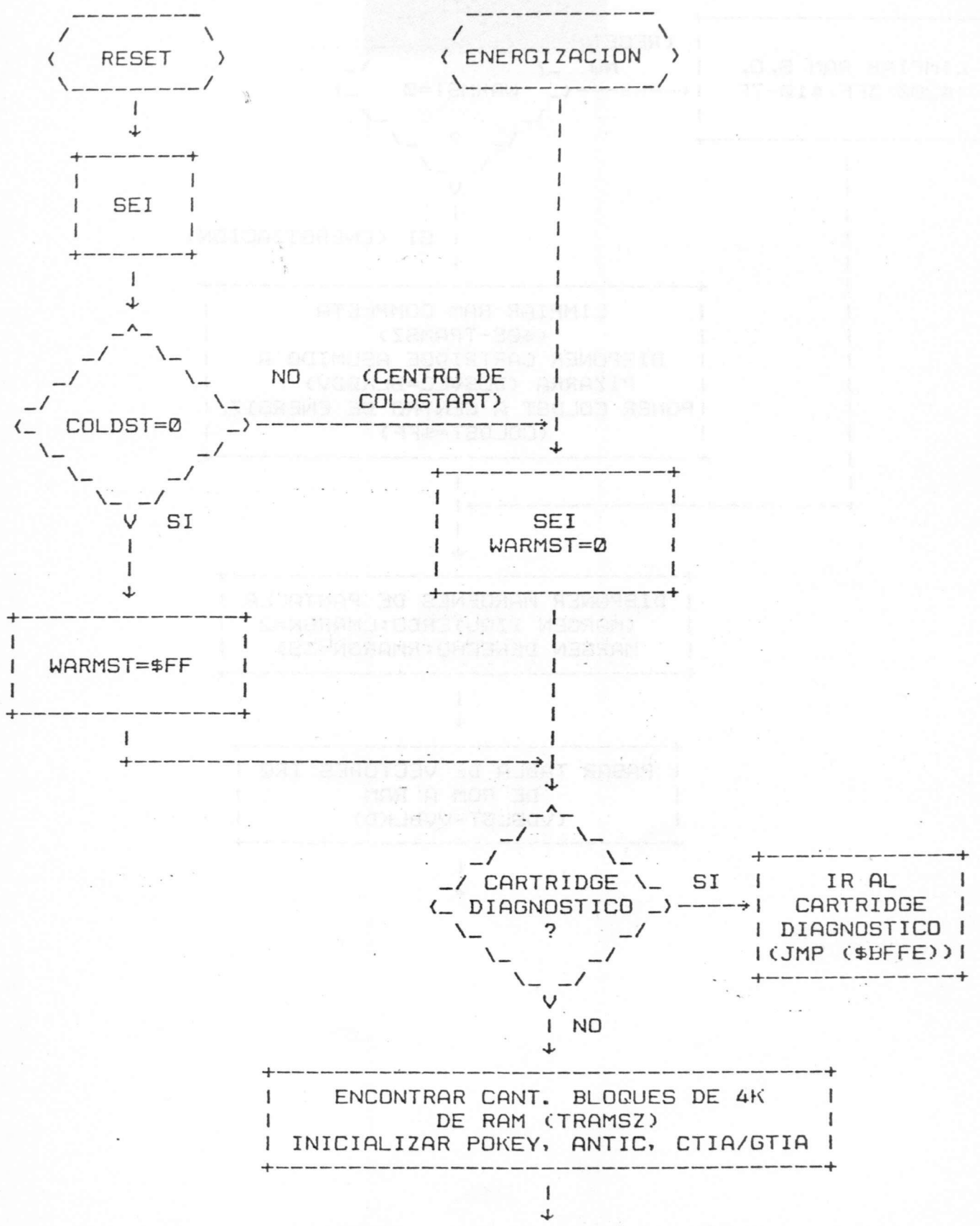
```
2370 OPEN #1,12,0,NOMARCH$
2375 GOSUB 9510
2380 RETURN
9500 REM 'PUNARCH.SUB'*WBB 19-03-81
9510 REM ESCRIBIR PUNTERO ARCHIVO
9515 S=ASC(PUNARCH$(1,1))*256+ASC(PUNARCH$(2,2))
9520 B=ASC(PUNARCH$(3,3))
9525 POINT #1,S,B
9530 PRINT #1;PUNARCH$
9535 RETURN
9600 REM DESPLEGAR PLANTILLA PANTALLA
9605 GRAPHICS 2
9610 PRINT #6;"* INICIANDO *"
9615 PRINT #6
9620 PRINT #6;"ACTUAL"
9625 PRINT #6;" TOTAL"
9630 PRINT #6;"% COMP"
9635 PRINT #6
9640 PRINT #6;"SECTOR"
9645 PRINT #6;"  BYTE"
9650 RETURN
9700 REM REFRESCAR PANTALLA
9705 PLOT 10,2:PRINT #6;I
9710 PLOT 10,3:PRINT #6;MAXARCH
9715 PLOT 10,4:PRINT #6;INT(I/MAXARCH*100)
9720 PLOT 10,6:PRINT #6;S;"    "
9725 PLOT 10,7:PRINT #6;B;"    "
9795 RETURN
9800 REM TRAP.SUB
9810 PRINT "ERROR ";PEEK(195);" EN ";PEEK(186)+256*PEEK(187)
9815 PRINT "CONFIRME ";
9820 INPUT SN$
9825 END
9850 REM 'RETARDO.SUB' WWB 19-03-81
9851 REM RETARDA LA EJECUCION EN 2,5 SEG
9852 REM (PARTIDA-P20)
9860 P20=PEEK(20)+150:IF P20>255 THEN P20=P20-256
9865 IF PEEK(20)<>P20 THEN 9865
9870 RETURN
```

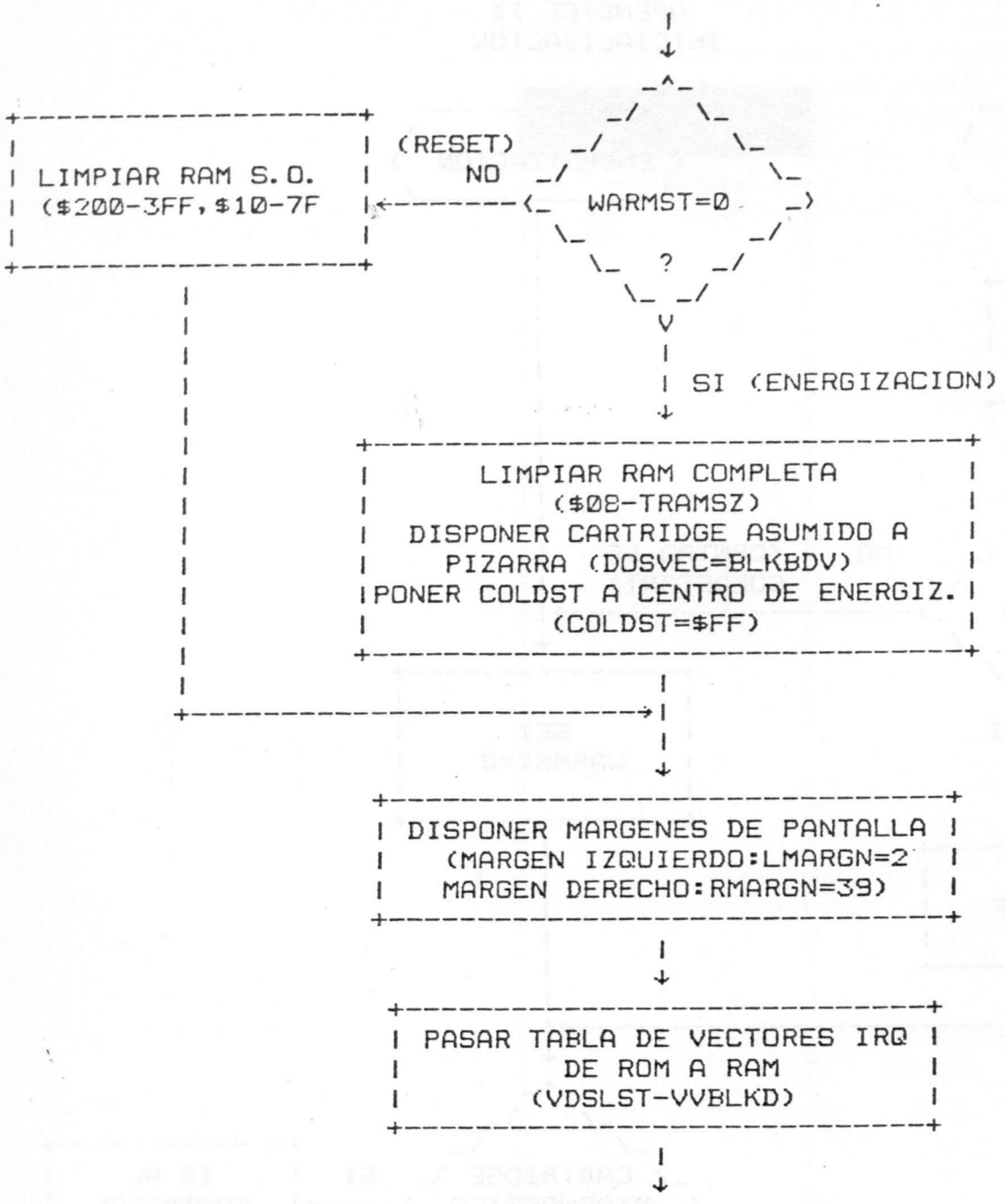
```
10 REM 'ARCHEX' WBB 31-03-81
100 REM EJEMPLO DE RUTINAS DE ACCESO ALEATORIO
101 REM ESTE PROGRAMA PROCESA EL ARCHIVO D1:CODAREA.DAT
102 REM DEBE CREARSE PREVIAMENTE CON 'ARCH0001'
103 REM CONSTA DE REGISTROS DE 24 BYTES c/u:
    1-3 CODIGO DE AREA, 4-24 DESC. UBICACION
104 REM
110 GRAPHICS 0
120 PRINT "'ARCHEX'":PRINT :PRINT "INICIALIZANDO"
200 REM INICIALIZAR VARIABLES
210 DIM ARCH$(15),CODA$(3),UBI$(21),SN$(1)
220 CANAL=1
230 ARCH$="D1:CODAREA.DAT"
300 REM ABRIR ARCHIVO DE DATOS
310 PRINT "ABRIENDO ARCHIVO DE DATOS"
320 GOSUB 9300
400 REM COMENZAR INGRESOS DE OPERADOR
410 PRINT
420 PRINT "(0=FIN) CODIGO DE AREA";:INPUT CODA$:
    IF CODA$="0" THEN 900
500 REM BUSQUEDA DE REGISTROS ACTIVOS CON ESTE CODIGO DE AREA
510 HAY=0
520 FOR REGISTRO=1 TO MAXARCH
530 GOSUB 9600
540 IF STS=1 THEN GOSUB 5000
550 NEXT REGISTRO
560 IF HAY=1 THEN 400
600 REM HOY NINGUNO, SE PUEDE AGREGAR
610 PRINT "NO HAY REGISTROS CON ESTE CODIGO"
620 PRINT "(S/N) QUIERE AGREGAR ";:INPUT SN$
630 IF SN$="S" THEN 400
700 REM AGREGAR LO SOLICITADO
710 GOSUB 9400
720 IF REGISTRO=0 THEN PRINT "ARCHIVO LLENO,
    NO SE PUEDE AGREGAR!":GOTO 400
730 PRINT "UBICACION: ";:INPUT UBI$
740 REGARCH$=CODA$
750 REGARCH$(4)=UBI$
800 REM PONER ERCHIVO AL DIA
810 GOSUB 9600
820 PRINT #CANAL;REGARCH$
830 GOSUB 9500
840 GOTO 400
900 REM ELIMINACIONES
910 PRINT
920 PRINT "(S/N QUIER ELIMINAR ALGO ";:INPUT SN$
930 IF SN$="S" THEN 1200
1000 REM ESPECIFICAR ELIMINACION
1010 PRINT
1020 PRINT "(0=FIN) CODIGO DE AREA ";:
     INPUT CODA$:IF CODA$="0" THEN 1200
```

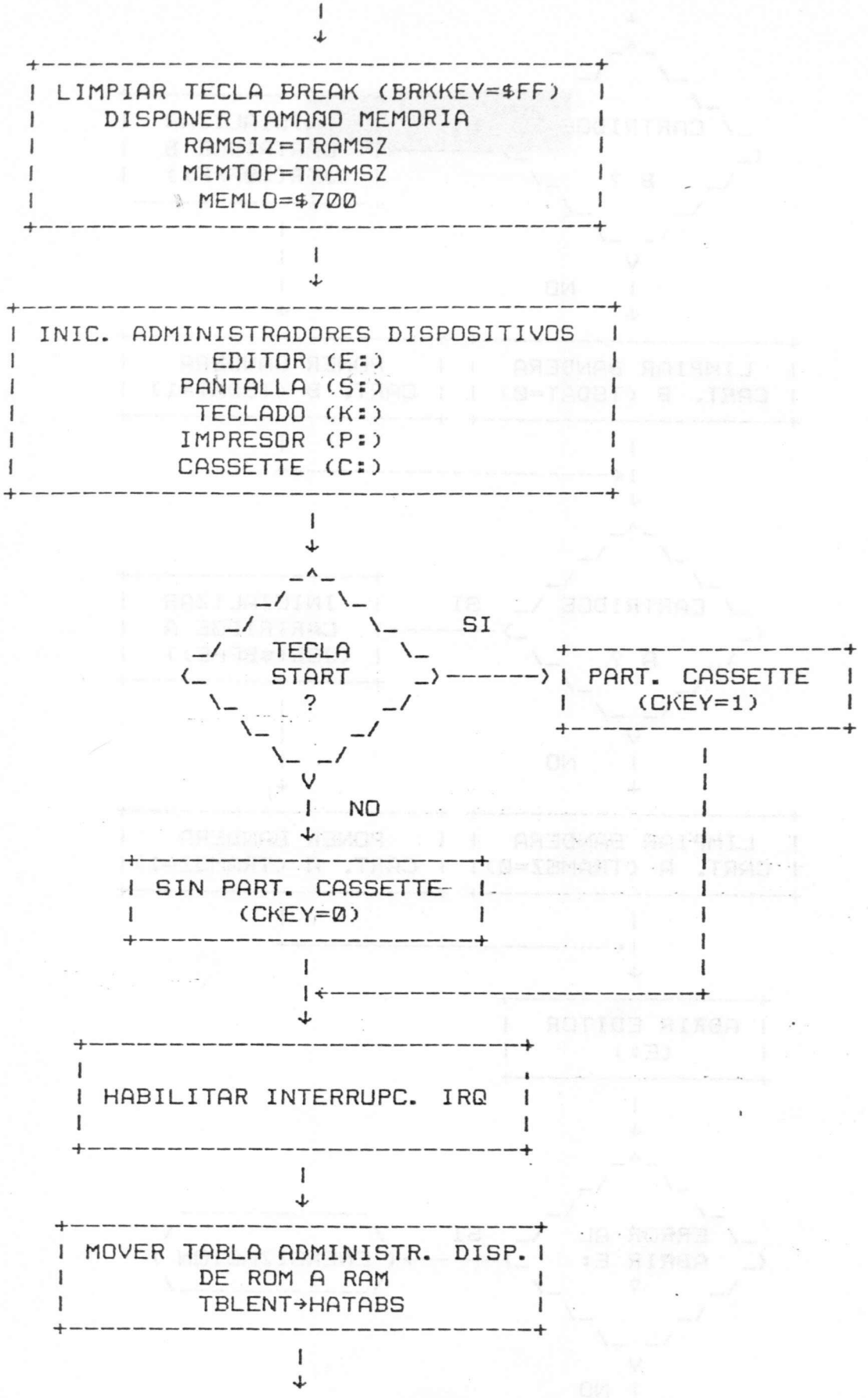
```
1100 REM BUSQUEDA DE REGISTROS ACTIVOS CON ESTE
     CODIGO DE AREA
1110 FOR REGISTRO=1 TO MAXARCH
1115 GOSUB 9600
1120 IF STS=1 THEN GOSUB 5100
1125 NEXT REGISTRO
1130 GOSUB 9500
1140 GOTO 1000
1200 REM IMPRIMIR ARCHIVO EN PANTALLA
1210 PRINT :PRINT "COD.","UBICACION":PRINT
1220 FOR REGISTRO=1 TO MAXARVH
1230 GOSUB 9600
1240 IF STS=1 THEN GOSUB 5200
1250 NEXT REGISTRO
1300 REM SALIDA POR IMPRESOR
1310 PRINT :PRINT "(S/N) QUIERE UD. UNA LISTA
     IMPRESA ";:INPUT SN$
1320 IF SN$="S" THEN 4900
1330 LPRINT "COD.","UBICACION":LPRINT
1340 FOR REGISTRO=1 TO MAXARCH
1350 GOSUB 9600
1360 IF STS=1 THEN GOSUB 5300
1370 NEXT REGISTRO
1380 GOTO 4900
4900 REM FIN
4910 CLOSE #CANAL
4920 PRINT "FIN DE LA EJECUCION"
4930 END
5000 REM PROCESAR REGISTROS ACTIVOS/DESPLEGAR
5010 INPUT #CANAL,REGARCH$
5020 IF REGARCH$(1,3)<>CODA$ THEN RETURN
5030 HAY=1
5040 PRINT "UBICACION: ;REGARCH$(4)"
5050 RETURN
5100 REM PROCESAR REGISTROS ACTIVOS/ELIMINAR
5110 INPUT #CANAL,REGARCH$
5120 IF REGARCH$(1,3)<>CODA$ THEN RETURN
5130 GOSUB 9450
5150 PRINT "ELIMINADO ";REGARCH$(4)
5160 RETURN
5200 REM PROCESAR REGISTROS ACTIVOS/IMPRIMIR
5210 INPUT #CANAL,REGARCH$
5220 PRINT REGARCH$(1,1),REGARCH$(4)
5230 RETURN
5300 REM PROCESAR REGISTROS ACTIVOS/IMP.PAPEL
5310 INPUT #CANAL,REGARCH$
5320 LPRINT REGARCH$(1,3),REGARCH$(4)
5330 RETURN
9300 REM 'ABREARCH.SUB' WBB 30-03-81
9305 REM ABRIR UN ARCHIVO EN MODO 12 Y DEFINIR SUS VARIABLES
9310 OPEN #CANAL,12,0,ARCH$
9315 DIM ENCARCH$(124)
9320 INPUT #CANAL,ENCARCH$
```

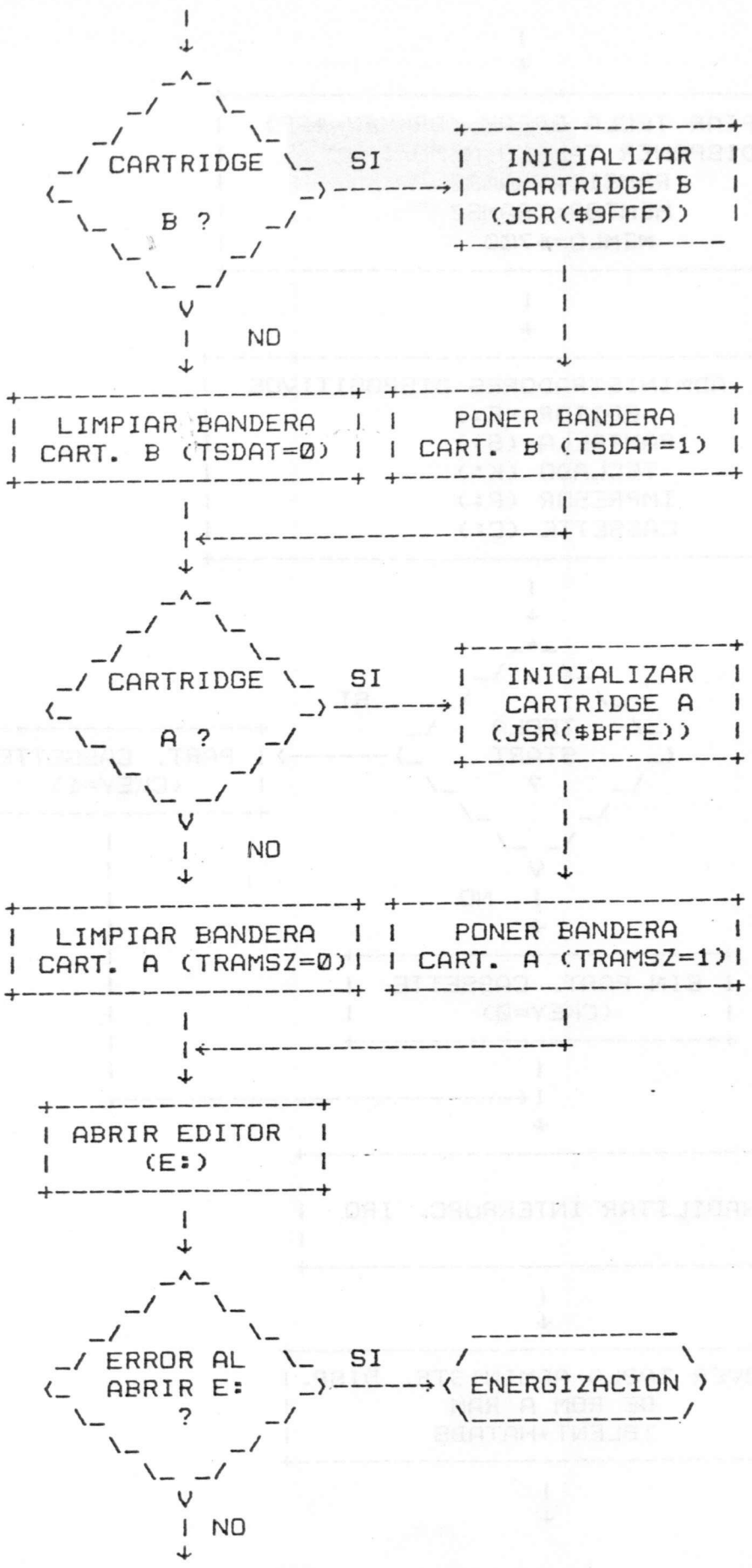
```
9325 SECARCH=ASC(ENCARCH$(1))*256+ASC(ENCARCH$(2))
9330 BYTARCH=ASC(ENCARCH$(3))
9335 STSARCH=ASC(ENCARCH$(4))
9340 MAXARCH=ASC(ENCARCH$(5))*256+ASC(ENCARCH$(6))
9345 LARARCH=ASC(ENCARCH$(7))*256+ASC(ENCARCH$(8))
9350 DIM PUNARCH$(4+4*MAXARCH),REGARCH$(LARACH)
9355 INPUT #CANAL,PUNARCH$
9360 RETURN
9400 REM 'AGRARCH.SUB' WBB 30-03-81
9405 REM ASIGNAR EL PROXIMO REGISTRO DISPONIBLE
9410 REGISTRO=0
9415 IF MAXARCH=0 THEN RETURN
9420 FOR REGISTRO1=1 TO MAXARCH
9425 B=REGISTRO1*4+4
9430 IF PUNARCH$(B,B)=CHR$(0) THEN
     REGISTRO=REGISTRO1:REGISTRO1=MAXARCH:
     PUNARCH$(B,B)=CHR$(1)
9435 NEXT REGISTRO1
9450 REM 'ELIARCH.SUB' WBB 30-03-81
9455 REM ELIMINAR UN ARCHIVO ACTIVO
9460 B=REGISTRO*4+4
9465 PUNARCH$(B,B)=CHR$(0)
9470 RETURN
9500 REM PUNARCH.SUB' WBB 19-03-81
9510 REM ESCRIBIR PUNARCH$
9515 S=ASC(PUNARCH$(1))*256+ASC(PUNARCH$(2))
9520 B=ASC(PUNARCH$(3))
9525 POINT #1,S,B
9530 PRINT #1;PUNARCH$
9535 RETURN
9600 REM 'UBIARCH.SUB' WBB 31-03-81
9605 REM APUNTAR ARCHIVO AL REGISTRO
9610 S=ASC(PUNARCH$(REGISTRO*4+1))*256+
     ASC(PUNARCH$(REGISTRO*4+2))
9615 B=ASC(PUNARCH$(REGISTRO*4+3))
9620 ATA=ASC(PUNARCH$(REGISTRO*4+4))
9625 POINT #CANAL,S,B
9630 RETURN
```

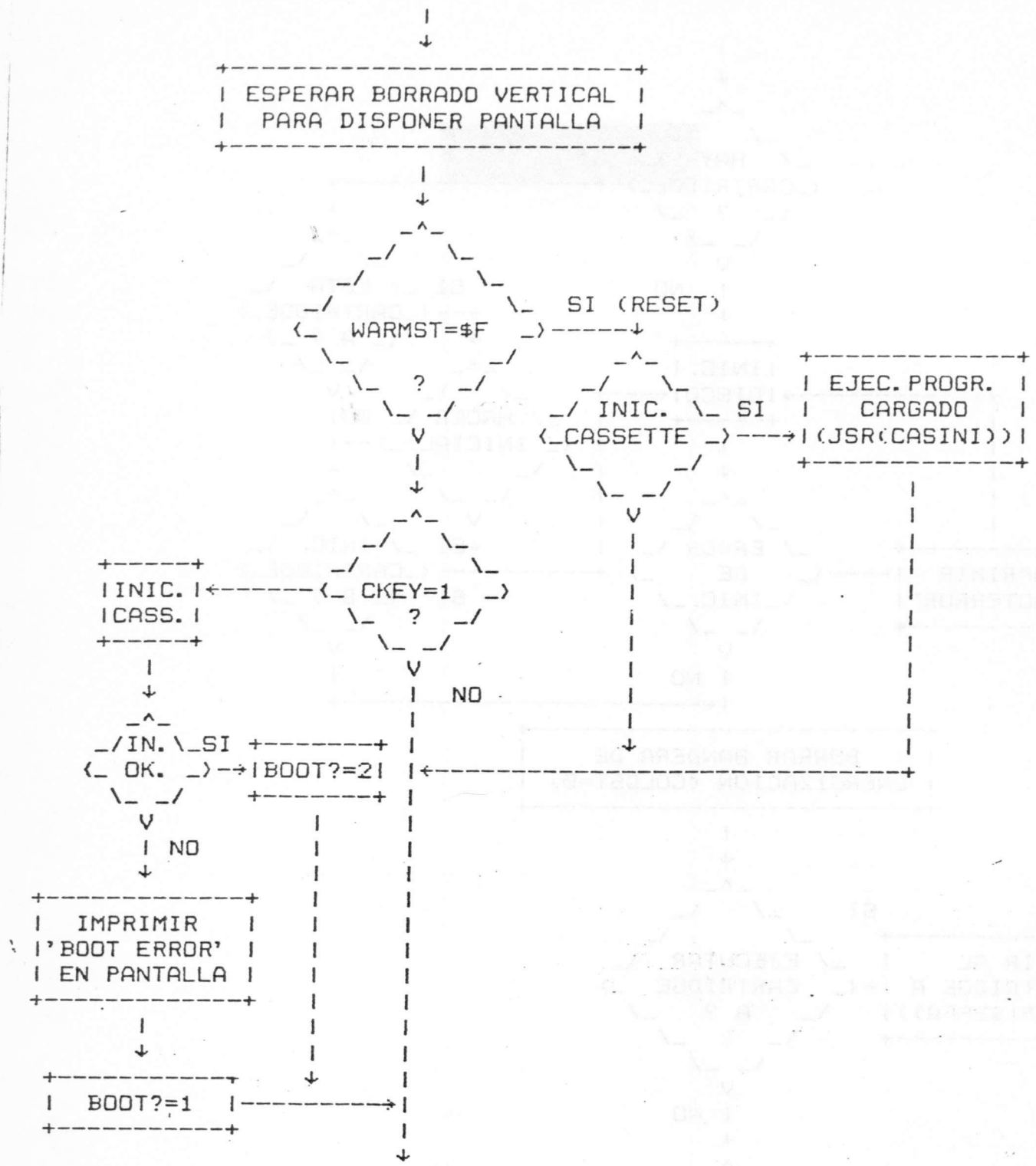
APENDICE IX
INICIALIZACION

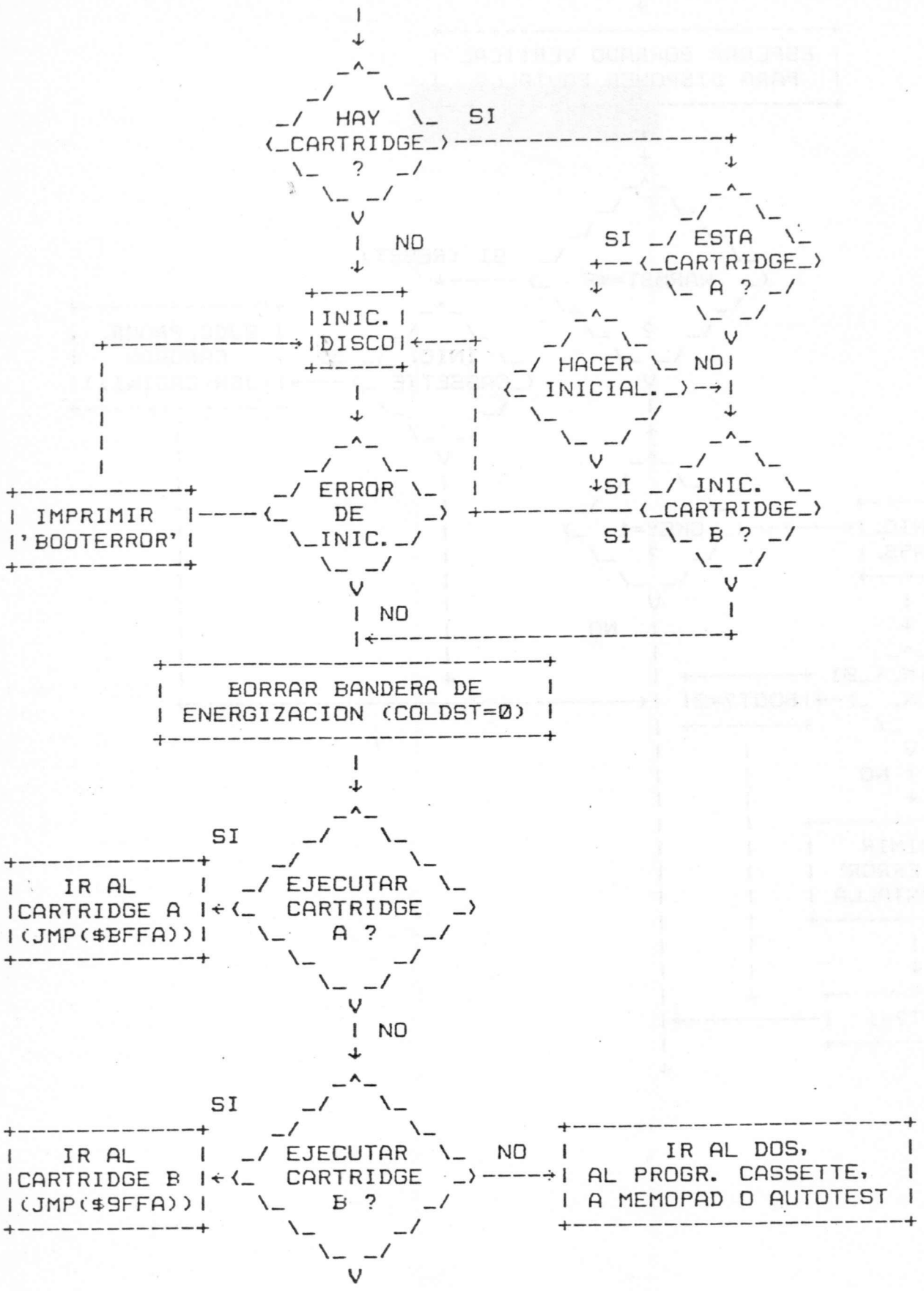



(RESET)
NO
WARMST=0
?
SI (ENERGIZACION)
LIMPIAR RAM S.O.
($200-3FF,$10-7F
LIMPIAR RAM COMPLETA
($08-TRAMSZ)
DISPONER CARTRIDGE ASUMIDO A
PIZARRA (DOSVEC=BLKBDV)
PONER COLDST A CENTRO DE ENERGIZ.
(COLDST=$FF)
DISPONER MARGENES DE PANTALLA
(MARGEN IZQUIERDO:LMARGN=2
MARGEN DERECHO:RMARGN=39)
PASAR TABLA DE VECTORES IRQ
DE ROM A RAM
(VDSLST-VVBLKD)


LIMPIAR TECLA BREAK (BRKKEY=$FF)
DISPONER TAMAÑO MEMORIA
RAMSIZ=TRAMSZ
MEMTOP=TRAMSZ
MEMLO=$700
INIC. ADMINISTRADORES DISPOSITIVOS
EDITOR (E:)
PANTALLA (S:)
TECLADO (K:)
IMPRESOR (P:)
CASSETTE (C:)
TECLA START ?
SI
PART. CASSETTE (CKEY=1)
NO
SIN PART. CASSETTE (CKEY=0)
HABILITAR INTERRUPC. IRQ
MOVER TABLA ADMINISTR. DISP.
DE ROM A RAM
TBLENT→HATABS


CARTRIDGE B ?
SI
INICIALIZAR CARTRIDGE B (JSR($9FFF))
NO
LIMPIAR BANDERA CART. B (TSDAT=Ø)
PONER BANDERA CART. B (TSDAT=1)
CARTRIDGE A ?
SI
INICIALIZAR CARTRIDGE A (JSR($BFFE))
NO
LIMPIAR BANDERA CART. A (TRAMSZ=Ø)
PONER BANDERA CART. A (TRAMSZ=1)
ABRIR EDITOR (E:)
ERROR AL ABRIR E:
SI
ENERGIZACION
NO


ESPERAR BORRADO VERTICAL
PARA DISPONER PANTALLA
WARMST=$F
SI (RESET)
?
INIC. CASSETTE
SI
EJEC. PROGR. CARGADO
(JSR(CASINI))
CKEY=1
?
INIC. CASS.
NO
IN. OK.
SI
BOOT?=2
NO
IMPRIMIR 'BOOT ERROR' EN PANTALLA
BOOT?=1


HAY CARTRIDGE ?
SI
NO
ESTA CARTRIDGE A ?
SI
HACER INICIAL.
NO
INIC. DISCO
ERROR DE INIC.
IMPRIMIR 'BOOTERROR'
INIC. CARTRIDGE B ?
SI
NO
BORRAR BANDERA DE ENERGIZACION (COLDST=Ø)
EJECUTAR CARTRIDGE A ?
SI
IR AL CARTRIDGE A (JMP($BFFA))
NO
EJECUTAR CARTRIDGE B ?
SI
IR AL CARTRIDGE B (JMP($9FFA))
NO
IR AL DOS, AL PROGR. CASSETTE, A MEMOPAD O AUTOTEST

APENDICE X
GTIA

GTIA es el ACTUAL circuito integrado de despliegue, que reemplazó al primitivo CTIA. De hecho, no es nada más que un CTIA con algunas características adicionales. Simplemente provee tres nuevos modos de interpretación a la información proveniente del circuito integrado ANTIC. ANTIC no requiere un modo nuevo para comunicarse con GTIA; al contrario, recurre al modo de alta resolución $F. GTIA, como ampliación, es totalmente compatible con CTIA. Sigue a continuación un breve resumen de las características de CTIA, para poder presentar en mejor forma las diferencias entre CTIA y GTIA.

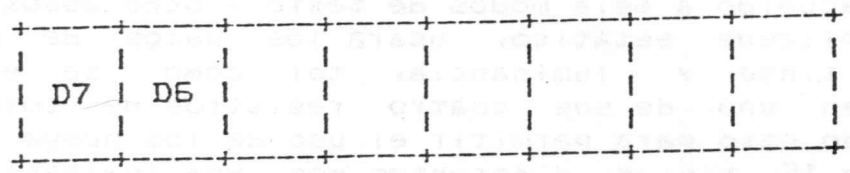
CTIA se diseñó para desplegar datos en la pantalla del televisor. Despliega los campos, los jugadores y los proyectiles y detecta translapos o colisiones entre objetos en la pantalla. CTIA interpretará los datos entregados por ANTIC, de acuerdo a seis modos de texto y ocho modos gráficos. En un despliegue estático, usará los datos de ANTIC para desplegar tinte y luminancia, tal como se encuentran definidos en uno de sus cuatro registros de color. GTIA expande todo esto para permitir el uso de los nueve registros de color o 16 tintes diferentes con una luminancia o 16 luminancias con un tinte en despliegue estático.

Los tres modos gráficos adicionales de GTIA son simplemente nuevas interpretaciones del modo ANTIC $F, un modo de alta resolución. Los tres modos afectan solamente al campo. Pueden agregarse siempre jugadores y proyectiles para introducir nuevos tintes o luminancias o para usar los mismos colores y luminancias en más de una forma. Todas las luminancias y los tintes desplegados pueden cambiarse al vuelo con interrupciones de lista de despliegue. GTIA usa 4 bits de datos de ANTIC para cada pixel, llamados datos de pixel. Cada pixel tiene un ancho de 2 compases de color y un alto de una línea de barrido. Así los pixels son aproximadamente 4 veces más anchos que altos. El despliegue tiene una resolución de 80 pixels horizontales por 192 verticales. Cada línea por tanto requiere 320 bits o 40 bytes de memoria, el mismo número de bytes requerido por ANTIC para su modo $F. Por ello, para que un programa pueda ejecutarse bajo modos GTIA, debe tener al menos 8 Kbytes de RAM libres para el despliegue.

Los modos GTIA se seleccionan por el registro de prioridad PRIOR. PRIOR tiene su copia en la ubicación $26F hexadecimal del Sistema Operativo y se encuentra en $D01B hexadecimal en el circuito integrado. Los bits 6 y 7 son los bits de control. Cuando ninguno de los dos se encuentra puesto, no se ha establecido modo GTIA y por lo tanto GTIA se comporta como CTIA. Cuando D7 es 0 y D6 es 1 se especifica el

modo 9, que permite la presentación de 16 luminancas diferentes del mismo tinte. Recuerde que los datos del pixel suministrados por ANTIC tienen 4 bits de ancho, lo que significa que pueden representarse 16 valores diferentes. Los jugadores y proyectiles pueden usarse en este modo para introducir tintes adicionales. Cuando D7 es 1 y D6 es 0 se especifica el modo 10. Este modo entrega 9 colores en el despliegue, usando los cuatro registros de color de campo y adiconalmente los cuatro registros de color de jugador proyectil y el registro de color de fondo. Cuando se usan jugadores en este modo, los cuatro registros de color jugador/proyectil también se usan para ellos. Cuando D7 es 1 y D6 es 1, se especifica el modo 11. Este modo entrega 16 tintes, todos con la misma luminancia, ya que hay 16 valores diferentes posibles de representar con cuatro bits. Jugadores y proyectiles pueden usarse en este modo para introducir luminancias diferentes.

PRIOR



D7	D6	OPCION	
0	0	No es modo GTIA (operacion en modo CTIA)	(modos 0-8)
0	1	1 tinte, 16 luminancias	(modo 9)
1	0	9 tintes+luminancias	(modo 10)
1	1	16 tintes, 1 luminancia	(modo 11)

Figura X.1
Los bits de PRIOR que seleccionan GTIA

Disponer los modos GTIA es igual de simple como disponer los modos CTIA. Se implementa desde BASIC, usando los comandos GRAPHICS 9, GRAPHICS 10 y GRAPHICS 11 para los modos 9, 10 y 11 respectivamente. En lenguaje assembler, la selección de uno de estos modos es idéntico al abrir la pantalla a cualquiera de los otros modos. Si Ud. construye su propia lista de despliegue, debe disponer PRIOR para escoger el modo correcto de acuerdo a Figura X.1

El modo 9 produce hasta 16 luminancias diferentes del mismo tinte. ANTIC entrega los datos de pixel que selecciona cada una de las 16 diferentes luminancias. El registro de color de fondo provee el tinte. En BASIC se logra esto, usando el comando SETCOLOR para disponer el color de tinte en

el par de bits (nybble) superior del registro de color de fondo y para disponer el valor de luminancia, el par de bits inferior en 0. El formato del comando es :

SETCOLOR 4, valor de tinte, 0

en que 4 especifica el registro de color de fondo. El "valor de tinte" dispone el tinte y puede tener cualquier valor entre 0 y 15; el 0 corresponde a la parte de luminancia del registro que se pone en 0. Debe hacerse esto, ya que los datos de pixel de ANTIC se combinarán con un "OR" lógico con el par de bits inferior del registro de color de fondo para disponer la luminancia que aparecerá sobre la pantalla. A continuación se usa el comando COLOR para seleccionar la luminancia del trazado en la pantalla, usando valores comprendidos entre 0 y 15 para este parámetro. Así, un programa BASIC incluirá al menos las siguientes sentencias, para usar el modo 9:

```
GRAPHICS 9              Para especificar el modo 9.
SETCOLOR 4,12,0         Para inicializar el registro de
                        color de fondo a algún valor de
                        tinte, en este caso verde.
FOR I=0 TO 15           Alguna forma de usar
COLOR I                 el comando COLOR para
PLOT 4,I+10             variar la luminancia.
NEXT I
```

En lenguaje Assembler use la copia del sistema operativo para el registro de color de fondo $2C8, para disponer el tinte en los cuatro bits superiores con valores hexadecimales comprendidos entre $0 y $F. Si se usan llamados al CIO, almacene los datos de pixel en el registro del sistema operativo ATACHR ubicado en $2FB. Ello selecciona la luminancia con valores hexadecimales comprendidos entre $0 y $F. Si Ud. mantiene sus propios datos de despliegue, los datos de pixels irán directamente a la mitad izquierda o derecha del byte de despliegue de RAM.

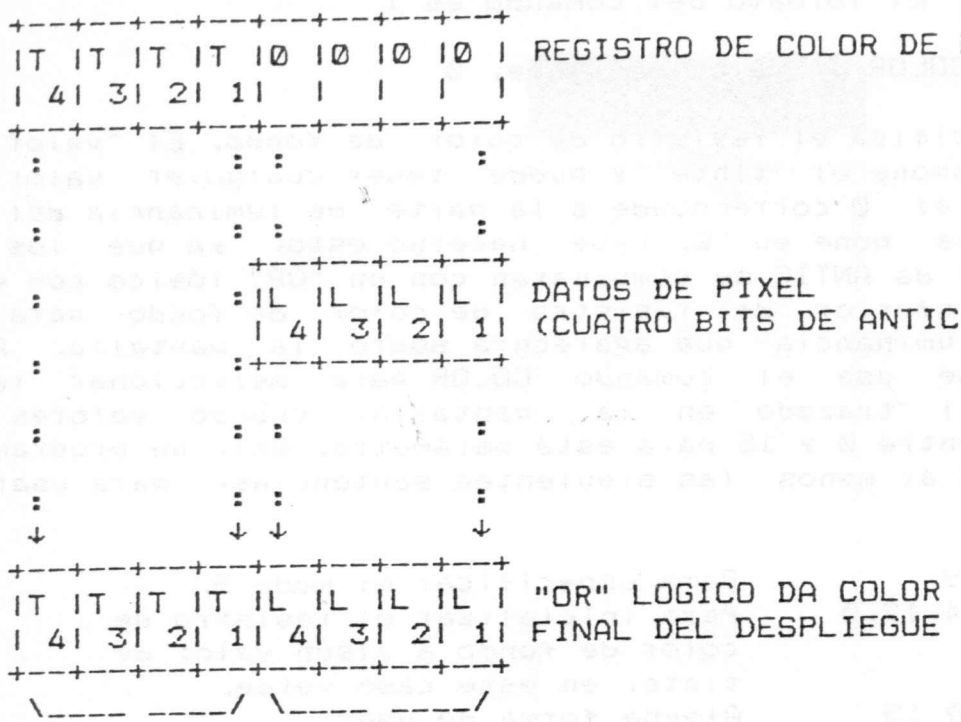


Figura X.2
"OR" del registro de color de fondo con los datos del pixel para dar el color final.

El modo 11 es similar al modo 9, excepto en que provee 16 tintes diferentes, todos ellos de la misma luminancia. Nuevamente ANTIC proveerá los datos de pixel para seleccionar, ahora 1 de 16 tintes diferentes. En BASIC se usa el comando SETCOLOR para disponer el valor de luminancia en el par de bits inferior del registro de color de fondo, mientras que en el par superior el valor de tinte se pondrá en 0. El formato del comando es

SETCOLOR 4,0,valor de luminancia

en que el 4 especifica el registro de color de fondo, el 0 pone el par de bits superior en 0 y el "valor de luminancia" dispone el valor de la luminancia, pudiendo variar entre 0 y 15. Al igual que en los demás modos gráficos (excepto el modo 9), el primer bit de la luminancia no se usa, de modo que en términos efectivos, solamente los números pares dan luminancias

diferentes, lo que arroja un total de 8 luminancias diferentes posibles en este modo. El comando COLOR se usa en este modo para seleccionar los diferentes tintes, usando valores de 0 a 15 para este parámetro. Los datos de pixel de ANTIC se combinarán por "OR" lógico con el par de bits superior del registro de color de fondo, para dar la parte de tinte del valor que en último término genera el color en la pantalla. Así, un programa BASIC que use el modo 11, incluirá al menos las siguientes sentencias:

```
GRAPHICS 11          Para especificar el modo 11.
SETCOLOR 4,0,12      Para inicializar el registro.
                     de color de fondo a alguna lumi-
                     nancia, muy brillante en este caso.
FOR I=0 TO 15        Alguna forma de usar el comando
COLOR I              COLOR, para variar el tinte.
PLOT 4,I+10
NEXT I
```

En lenguaje Assembler, recurra a la copia del sistema operativo para el registro de color de fondo en $2C8, para disponer la luminancia en los 4 bits inferiores con un valor hexadecimal entre 0 y $F. Si se usan llamados CIO, almacene los datos de pixels en ATACHR ubicado en $2FB. Con ello se selecciona un tinte con valores comprendidos entre 0 y $F. Si Ud. mantiene sus propios datos de despliegue, los datos del pixel irán directamente a la mitad izquierda o derecha del byte de despliegue RAM.

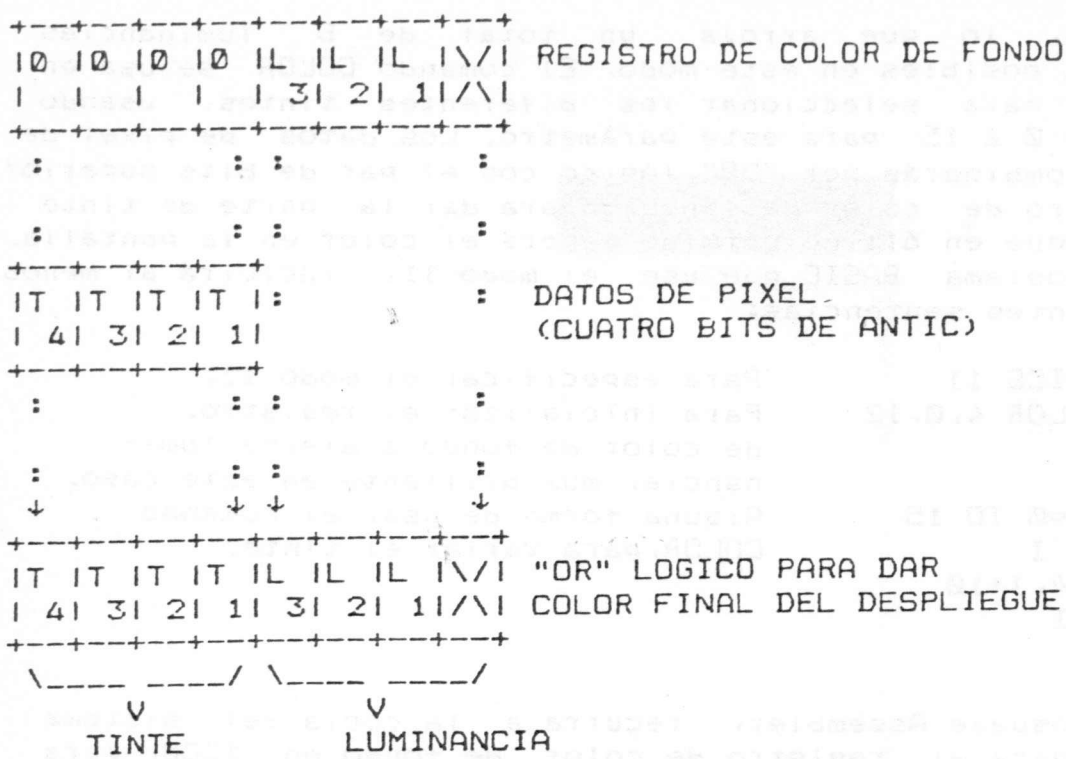


Figura X.3
El "OR" del registro de color de fondo con el pixel de datos para dar el color final

El modo 10 permite que todos los nueve registros de color se usen en el campo de una vez. Cada registro de color que se use debe disponerse para alguna combinación de tinte y luminancia. El dato de pixel de ANTIC se usa en este modo para seleccionar uno de los registros de color para el despliegue. En BASIC puede usarse el comando SETCOLOR, tal como se describe en el manual de referencia BASIC, para disponer los colores de fondo y los registros de los cuatro campos. Ellos también pueden disponerse usando la instrucción POKE a las direcciones 708 a 712, donde están ubicados los registros de los cuatro campos y el registro del fondo. La instrucción POKE debe usarse forzosamente para disponer los registros de color de jugador-proyectil en las ubicaciones 704 a 707. Se usa el comando COLOR para seleccionar el registro de color deseado. Los unicos valores con sentido para sus argumentos son 0 a 8. Nace un problema con este modo. ANTIC entrega 4 bits de datos por pixel, tal como lo hace en los modos 9 y 11. Esto permitiría la selección de 16 registros de color. Sin embargo, solamente existen 9

registros de color en el circuito. Valores de datos ilegales comprendidos entre 9 y 15 no harán más que seleccionar un valor inferior de registro de color. Un programa BASIC que haga uso del modo 10 incluirá:

1) un comando GRAPHICS 10 para especificar el modo 10;
2) un conjunto de instrucciones POKE para disponer tintes y luminancias en los registros de color,
   o una combinación de comandos SETCOLOR e instrucciones POKE para hacerlo;
3) un comando COLOR para seleccionar el registro de color deseado.

El lenguaje Assembler almacena los datos de pixel en ATACHR ($2FB) o directamente en el byte de RAM de despliegue, tal como en los modos 9 y 11. En este modo los datos de los pixels pueden alcanzar de 0 a 8 y seleccionar uno de nueve registros de color.

VALOR SENTENCIA COLOR	REGISTRO DE COLOR USADO	COPIA S.O.
0	D012	2C0
1	D013	2C1
2	D014	2C2
3	D015	2C3
4	D016	2C4
5	D017	2C5
6	D018	2C6
7	D019	2C7
8	D01A	2C8

Figura X.4
Referencia de números y ubicaciones de registros de color y comandos COLOR.

Hay un punto importante en relación a GTIA, que se refiere a la compatibilidad. GTIA es totalmente compatible con CTIA en el sentido que puede ejecutar todas las instrucciones CTIA y que todos los programas que funcionan en sistemas CTIA, funcionan exactamente igual con GTIA. Esto significa que siempre se dispone de todo el uso de jugadores y proyectiles, se dispone de detección de colisión y de traslapo y de interrupciones de lista de despliegue. Los modos gráficos GTIA son totalmente soportados por el sistema operativo y los comandos gráficos y los utilitarios que ejecutan en los modos CTIA pueden usarse en los modos GTIA.

Se dispone de más colores de una vez en el despliegue de la pantalla. Pueden ocurrir dieciseis cambios de color en una línea en forma totalmente independiente de la intervención del procesador. Esto de hecho es más de lo que puede hacerse con interrupciones de lista de despliegue que darían como máximo solamente 12 cambios de color por línea. Se pueden representar contornos más finos y profundidad usando el sombreado disponible en el modo 9. Esto significa que pueden presentarse gráficos tridimensionales con realismo.

Por otro lado hay algunas desventajas: los modos GTIA son modos de mapa, por lo cual no puede desplegarse texto en estos tres modos. Debe usarse una lista de despliegue a propósito para conmutar a un modo que soporte despliegue de caracteres. El pixel GTIA es un rectángulo horizontal, largo y angosto (relación ancho a alto 4:1) y no presenta muy bien las líneas curvas. Como cada pixel usa 4 bits de información, GTIA requiere cerca de 8 K de RAM libre para funcionar. Aunque los programas CTIA funcionan en GTIA, la compatibilidad inversa no es cierta. Los modos GTIA no producirán despliegues correctos en computadores que tienen CTIA. Podrán resultar reconocibles, pero no tan coloridos. Por el momento un programa no puede reconocer si en el sistema se encuentra un GTIA o un CTIA.

GLOSARIO

ACCESO DIRECTO DE MEMORIA (DMA)

Ocurre cuando ANTIC o el 6502 acceden un byte de instrucción o un byte de datos desde la memoria.

ADMINISTRADORES DE DISPOSITIVO

Rutinas que se encuentran en la ROM del sistema operativo y que se llaman a través de CIO (siempre que el administrador tenga una entrada en HATABS) para la comunicación con un dispositivo particular. Actualmente se soportan: el editor de despliegue, la pantalla, el teclado, el impresor y el cassette. Un programa de aplicación puede agregar en RAM administradores de dispositivo adicionales, p. ej. él de la interfaz ATP-850, R:.

ADMINISTRADOR RESIDENTE DE DISCO

La programación que realiza cinco funciones de entrada/salida de disco de bajo nivel importantes que son: FORMATEAR, LEER SECTOR, ESCRIBIR SECTOR, ESCRIBIR/VERIFICAR SECTOR, Y STATUS.

ANIMACION CICLICA

Técnica de saltar en forma repetitiva a través de colores, imágenes gráficas o juegos de caracteres gráficos para animar imágenes de pantalla.

ANIMACION DE CAMPO

La técnica de animar un objeto, moviendo su imagen bidimensional de bytes a una ubicación nueva en la memoria de pantalla y a continuación borrando los bytes que definían la imagen anterior antes, de desplegar la imagen recién movida.

ANTIC

Un microprocesador independiente, comprendido dentro de los ATARI, dedicado al despliegue de televisión. ANTIC es programable por el usuario mediante un juego de instrucciones, un programa (la "lista de despliegue") y datos (la "memoria de despliegue").

ARCHIVO

En entrada/salida de cassette consiste de una guía de 20 segundos del tono de marca más algún número de byte de datos y un fin de archivo. En entrada/salida de diskette consiste de un número de sectores enlazados por punteros (125 bytes de datos por sector).

ARCHIVO DE INICIACION DE CASSETTE

Un archivo normal o creado por el usuario, que inicia el computador desde el cassette al energizar o al presionar SYSTEM RESET.

AREA DE AGRUPAMIENTOS STRING

En BASIC, este bloque contiene todos los datos de strings y de agrupamientos.

AREA DE JUGADOR-PROYECTIL

Un área de RAM que contiene las imágenes de los cuatro jugadores y cuatro proyectiles de una gráfica de jugador-proyectil, tanto como algo de RAM adicional. El área de jugador-proyectil debe estar en un límite de 1K para resolución de jugadores de una línea y en un límite de 2K para resolución de jugadores de doble línea.

ARTIFICIOS DE TELEVISION

Se refiere esto a un punto o "pixel" de la pantalla que despliega un color diferente al que se le ha asignado. Los "artificios" son posibles en los modos ANTIC 2, 3 y 15, que corresponden a los modos BASIC 0, ninguno, y 8.

BARRIDO DE TRAMA

Un sistema de despliegue de televisión, que recurre al haz de electrones generado en la parte posterior del tubo de imagen de televisión. El haz barre a través de la pantalla en una forma regular orientada de izquierda a derecha y de arriba hacia abajo.

BASE DE DATOS DEL SISTEMA

Es un área que ocupa en RAM las páginas 0 a 4 y que contiene muchas ubicaciones con información de interés para el usuario.

BIT DE HABILITACION DE DESLIZAMIENTO HORIZONTAL

Es el bit DE de la instrucción de despliegue ANTIC que habilita el deslizamiento horizontal a través del registro HSCROL.

BIT DE HABILITACION DE DESLIZAMIENTO VERTICAL

Es el bit D5 del byte de instrucción de la lista de despliegue ANTIC que habilita el deslizamiento fino vertical a través de VSCROL ($D405), el registro de deslizamiento vertical fino.

BIT-N

Un registro de status del procesador 6502 puestro, entre otros, por los llamados de entrada/salida (E/S), para indicar el éxito o fracaso de una operación de E/S.

BLOQUES DE CONTROL DEL SISTEMA ENTRADA/SALIDA

Estos bloques son elementos del subsistema de entrada/salida que se usan para comunicar información acerca de las funciones entrada/salida a realizarse.

BORDE

En el modo BASIC 0 es el área del despliegue de la pantalla de televisión que está formado por las cuatro orillas de la pantalla. El borde toma el color de fondo.

BORRADO HORIZONTAL

Es el perìodo durante el cual el haz de electrones (a medida que traza la imagen de pantalla) retorna del extremo derecho al extremo izquierdo de la pantalla.

BORRADO VERTICAL

Es el período durante el cual el haz de electrones (a medida que traza la imagen de pantalla) regresa desde la parte de abajo a la parte superior de la pantalla. Este período tiene una duración aproximada de 1400 microsegundos.

BRKKEY

Una bandera que pone el sistema operativo cuando detecta que se ha presionado la tecla BREAK. El valor normal de BRKKEY es $FF. Si cambia significa que se ha presionado la tecla BREAK.

BYTE DE CONTROL

En entrada/salida de cassette forma parte de cada registro. Contiene uno de tres valores posibles.

BYTE NOMBRE DE CARACTER

Un valor de memoria de despliegue de ANTIC de un byte que selecciona un carácter determinado dentro del juego de caracteres, según la posición numérica del carácter dentro de ese juego.

CAMPO AMPLIO

Una opción de ancho de despliegue de pantalla igual a 192 compases de color

CAMPO ANGOSTO

Una opción de ancho de despliegue de pantalla igual a un ancho de 128 compases de color.

CAMPO DE JUEGO

El área de la pantalla que se sobrepone directamente al fondo de la pantalla. Las gráficas de mapa y/o de texto forman este campo.

CAMPO NORMAL

Una opción de ancho de campo de despliegue igual al ancho de 160 compases de reloj.

CARACTER DE MARCA

Para entrada/salida de cassette es un valor 55(HEX) cuyo propósito es el ajuste de la tasa de baud. Incluyendo los bits de partida y de detención, cada carácter de marca tiene 10 bits de largo.

CHBAS

La ubicación de sombre del sistema operativo que usa ANTIC para encontrar el juego de caracteres a usar para los modos de despliegue de caracteres. CHBAS se encuentra en la dirección decimal 756.

CINTA MAESTRA

En la producción masiva de cintas cassette es ésta una cinta de carrette abierto de 1/4 de pista, 1/4 de pulgada y grabada a 7 1/2 pulgadas por segundos. La cinta maestra viene a ser el maestro fuente, previo al proceso de duplicación

CIO

Acrónimo de Rutina del Sistema Central de Entrada/Salida. (Central Input/Output System Routine). El CIO encamina los datos de control de entrada/salida al administrador de dispositivo correspondiente y a continuación entrega el control a ese administrador. CIO es al mismo tiempo el punto de entrada común para la mayoría de las funciones de entrada/salida del sistema operativo.

CODIGO (TOKEN)

En BASIC, un byte de 8 bits, que contiene un código de ejecución particular.

CODIFICACION

En BASIC, es el proceso de obtener una línea de ingreso de programa y crear a partir de ella una serie de bytes de 8 bits que contienen códigos con significado para la ejecución.

COLISION

Ocurre cuando la imagen de un jugador en la gráfica de jugador/proyectil coincide con otra imagen. Hay 60 colisiones posibles; a cada una de ellas se les ha asignado un bit que puede verificarse. Estos bits están colocados en 16 registros de GTIA (solamente se usan los 4 bits inferiores).

COLOR

Uno de los 128 valores obtenidos de una combinación tinte-luminosidad que se almacena en el registro de color.

COLOR DE JUGADOR

El color de un jugador en la gráfica de jugador-proyectil. Cada uno de los cuatro jugadores independientes tiene su propio color almacenado en el registro de color que le está asociado.

COLOR DE PROYECTIL

El color de un proyectil en la gráfica de jugador-proyectil. Cada uno de los cuatro proyectiles toma el color del jugador al que está asociado.

COLRSH

Una ubicación de página cero ($4F) dispuesta y vuelta a poner al día por el sistema operativo durante las interrupciones de borrado vertical para el procesamiento del modo de rotación. Cuando actúa el modo de rotación COLRSH recibe un nuevo valor aleatorio cada 4 segundos.

COMANDO

En BASIC, es éste el primer código ejecutable en una sentencia BASIC y que indica a BASIC que debe interpretar los códigos que siguen en una forma particular.

COMPAS DE COLOR

La unidad normal de distancia horizontal sobre la pantalla de televisión. Hay 228 compases de color en una línea de barrido horizontal.

CONSTANTE

En BASIC es un valor decimal codificado en binario de 6 bytes, precedido por un código especial. Este valor se mantiene inalterable a través de toda la ejecución del programa.

COPIA (SHADOWING)

El proceso por el cual una ubicación de software se usa por parte del sistema operativo para poner al día los registros circuitales durante los perìodos de borrado verticales.

CTIA

Un circuito integrado de interfaz de televisión controlado principalmente por ANTIC. CTIA convertìa los comandos digitales de ANTIC en una señal enviada al televisor. Reemplazado en la actualidad por GTIA.

CUENTA DE BYTE

Corresponde a la posición del puntero dentro del sector de un diskette.

DCB

Acrónimo de block de control de dispositivo. El DCB se usa en el subsistema entrada/salida para la comunicación entre los administradores de dispositivo y el SIO.

DESLIZAMIENTO DIAGONAL

Resulta de la combinación de los deslizamientos horizontal y vertical de una imagen de pantalla.

DESLIZAMIENTO FINO

El proceso de deslizar horizontalmente o verticalmente una imagen de pantalla en pasos de compás de reloj o de lìnea de barrido. Deben usarse los registros circuitales de desplizamiento horizontal y de deslizamiento vertical para lograr deslizamiento fino.

DESLIZAMIENTO GRUESO

El proceso de alterar los bytes de la dirección LMS (carga de barrido de memoria) de la lista de despliegue para deslizar vertical u horizontalmente la imagen de pantalla, un byte a la vez. Esto se logra agregando uno o restando uno de los bytes de dirección del LMS.

DESLIZAMIENTO HORIZONTAL

Es el proceso de escurrir una ventana de pantalla hacia la izquierda o derecha por sobre la memoria de despliegue para desplegar más información de la que podría verse con una pantalla estática. Se dispone tanto de deslizamiento grueso como de deslizamiento fino en el sentido horizontal.

DESLIZAMIENTO VERTICAL

El proceso de "rodar" una "ventana" de despliegue sobre un volúmen mayor de datos de pantalla de la memoria de despliegue del que puede desplegarse con una ventana de pantalla estática. El computador ATARI posee, tanto deslizamiento grueso como fino en el sentido vertical

DETECCION DE COLISION

Un valor para juegos principalmente. Es la habilidad incluìda en los circuitos, para detectar si ha ocurrido alguna de las 60 posibilidades de colisión jugador-proyectil.

DMACTL

El registro circuital cuya disposición de bits controla, entre otras cosas, la resolución vertical de un jugador y la habilitación de la gráfica de jugador-proyectil.

DOS

Acrónimo de Sistema Operativo de Disco, que es una extensión del Sistema Operativo en el sentido de permitir al usuario acceder al almacenamiento masivo de las unidades de disco como archivos.

DRKMSK

Una ubicación de página Ø ($4E), dispuesta y mantenida al día por el sistema operativo durante las interrupciones de borrado vertical para el procesamiento del modo de rotación. El uso de DRKMSK corta el bit de mayor valor de la luminancia de la cuaterna derecha del valor del registro de color. Esto asegura una luminancia baja del modo de rotación.

DUP

Acrónimo de paquete utilitario de disco. DUP está constituìdo por un conjunto de utilitarios para diskette conocido como el menú del DOS. El DUP ejecuta comandos, llamando el FMS a través del CIO.

EOL

En BASIC, "fin de línea", un carácter con el valor $9B.

ESPACIO

Para entrada/salida de cassette corresponde a una salida de 3995 HZ a la cinta de cassette como limitador en conjunto con los tonos de marca.

ESPECIFICACION DE DISPOSITIVO

Un código especial de HATABS que especifica un dispositivo de entrada/salida en particular.

ESTILO

Un conjunto de caracteres que constituyen un juego de caracteres. Estos caracteres pueden corresponder a imágenes de texto o de gráfica.

FMS

Sistema de manejo de archivos. Es un administrador de dispositivo no residente, que soporta algunas funciones especiales del CIO.

FONDO

El área del despliegue de pantalla de televisión sobre el cual se proyectan objetos generados por la gráfica de jugador-proyectil o los campos de juego. El fondo tiene su color definible por el usuario.

FORMATEO

Un comando del administrador de disco residente, que limpia todas las pistas del diskette.

FORMATO DE DESPLIEGUE

Una imagen de pantalla en papel que se traduce a una secuencia de líneas de modo, las que a su vez eventualmente se traducen a instrucciones de lista de despliegue ANTIC.

FUNCION

En BASIC, un código que al ser ejecutado retorna un valor al programa.

GRAFICA DE CARACTERES

Técnica de redefinir los caracteres individuales de un juego de caracteres para formar imágenes gráficas en vez de caracteres de texto..

HATABS

La tabla de puntos de entrada de los administradores de dispositivo que usa CIO. HATABS se encuentra en $031A.

HSCROL

Es el registro de deslizamiento horizontal fino, ubicado en $D404 y que contiene el número correspondiente a la cantidad de compases de color del deslizamiento horizontal de una línea.

IMAGEN DE CARACTER

La rejilla de pixels (8x8 en el modo gráfico 0 por ejemplo) que define la forma particular de cada carácter.

INDIRECCION DE JUEGO DE CARACTERES

La técnica de especificar un juego de caracteres particular para que lo use ANTIC, ubicando la dirección de página de comienzo de este juego en CHBAS.

INDIRECCION DE REGISTRO DE COLOR

La técnica de especificar un color particular, codificando su valor en un registro de color.

INDIRECCION GRAFICA

Una característica especial del Sistema ATARI, que permite una generalidad de registros de color y juegos de caracteres, usando punteros indirectos a los valores de los juegos de caracteres y colores.

INTERRUPCION DE BORRADO VERTICAL

Una interrupción no-enmascarable que ocurre cada 60 avo de segundo durante el período de borrado vertical del despliegue de televisión. Durante esta interrupción, el sistema operativo realiza varias funciones domésticas, tales como la copia de los registros de color.

INTERRUPCION DE LISTA DE DESPLIEGUE

Una instrucción especial de lista de despliegue ANTIC que interrumpe al microprocesador 6502 durante el trazado de una imagen de pantalla, y que permite que el 6502 altere los parámetros de la pantalla.

INTERRUPCIONES SIO

Hay tres interrupciones IRQ que emplea el SIO para enviar y recibir comunicaciones de bus serial a los dispositivos del bus serial. Las tres son: VSERIR (entrada serial lista), VSEROR (requerimiento de salida de dial) y VSEROC (fin de transmisión).

INTERVALO INTERREGISTRO

Para registros entrada/salida de cassette, consiste del intervalo post-registro de un registro determinado, seguido del tono de escritura pre-registro del registro siguiente.

INTERVALO POST-REGISTRO

Una frecuencia pura de tono de marca, que se usa como limitador pre-registro en la entrada/salida de cassette.

IOCB

Acrónimo de bloque de control de entrada/salida. Hay 8 de estos bloques, cuya función es la comunicación entre los programas de usuario y el CIO.

IRQ

Interrupción enmascarable (puede ser habilitada o inhibida por el 6502) tales como un IRQ de tecla BREAK.

IRQEN

El registro de escritura solamente, que contiene los bits de habilitación - inhibición de IRQ. Hay una copia de IRQEN en POKMSK.

JUEGO DE CARACTERES REDEFINIDOS

Cualquier juego de caracteres definidos por usuario, designados para ser usados por ANTIC y que sea diferente al juego de caracteres en ROM. CHBAS (dirección decimal 756) contiene la dirección de la primera página de cualquier juego de caracteres redefinidos.

JUEGO NORMAL DE CARACTERES

El juego de caracteres asumido residente en ROM que usan los sistemas de computación ATARI.

JUGADOR

Una imagen unidimensional en RAM, usda en la gráfica de jugador-proyectil y que puede tener 128 bytes (resolución de 2 líneas) o 256 bytes (resolución de 1 línea) de largo. El jugador aparece como una franja vertical de 8 pixels de ancho, que se extiende desde la parte superior hasta la inferior de la pantalla. Hay un máximo de 4 jugadores independientes.

KERNEL O NUCLEO

Una primitiva técnica de programa - circuito, que consiste en un bucle de programa del 6502, ajustado precisamente en tiempo al ciclo de despliegue del televisor. El código de núcleo supervisa el registro VCOUNT u consulta las tablas de cambios de pantalla catalogadas como función de valores de VCOUNT, de modo que el 6502 pueda controlar arbitrariamente todos los valores gráficos para toda la pantalla.

LINEA

En BASIC, una línea consiste de una o más sentencias BASIC precedidas por un número de línea en el rango de 0 a 32767 o, si es una línea de modo inmediato, sin número de línea.

LINEA DE BARRIDO HORIZONTAL

La unidad de medida fundamental de distancia vertical sobre la pantalla. La línea de barrido está formada por un trazo del haz de electrones a través de la pantalla.

LISTA DE DESPLIEGUE

El programa de ANTIC definido por el usuario o dispuesto automáticamente (a través de un comando GRRAPHICS) POR BASIC. La lista de despliegue especifica dónde se encuentran los datos de pantalla, qué modos de despliegue deben usarse para interpretar los datos de pantalla y qué opciones especiales (si es que hay alguna) deben implementarse.

LISTA DE DESPLIEGUE DINAMICA

Es ésta una lista de despliegue ANTIC, la cual altera el 6502 durante los períodos de borrado verticales, permitiendo así un grado aún mayor de flexibilidad en el despliegue de pantalla.

LINEA DE MODO

Un grupo de líneas de barrido horizontal para despliegues de pantalla. Dependiendo del modo de despliegue BASIC o ANTIC en uso, una línea de modo estará compuesta por un número variable de líneas de barrido. Por la misma razón, dependiendo siempre del modo de despliegue, una imagen de pantalla se compondrá de un número variable de líneas de modo.

LOMEM

En BASIC, es éste el puntero ([80,81]hexadec.) a la memoria de transferencia usada para codificar una línea de código. Esta memoria de transferencia es de 256 bytes y se encuentra al final de la RAM asignada al sistema operativo.

LUMINANCIA

La cuaterna de bits inferior del color de un registro de color. Hay 8 números pares de valores de luminancias ($0 a $F, solamente valores pares) que en combinación con los valores de tinte producen los 128 colores disponibles en los ATARI.

MAESTRO DE TRABAJO

En la producción masiva de cintas cassette, ésta es la cinta maestra final a partir de la cual se fabricará una cantidad grande de cintas cassette.

MAESTRO INTERMEDIO

En la producción masiva de cintas cassette, es la copia de respaldo de la cinta de trabajo maestra.

MAESTRO FUENTE

En la producción masiva de cintas cassette, es lo mismo que la cinta maestra.

MARCA

Para entrada/salida de cassette, corresponde a la frecuencia de 5327 HZ.

MARCA DE SINCRONISMO

Es ésta una frecuencia de espacios: 3995 HZ, que se usa como una especie de marca de "fin de registro" para pistas de audio en el cassettte. En software de aplicación es útil para sincronizar el despliegue de pantalla del computador con el audio del cassette.

MEMORIA DE PANTALLA

Un área de RAM usada por el 6502 para almacenar los bytes de datos que recojerá ANTIC (por DMA, acceso directo de memoria) para interpretarlos y eventualmente desplegarlos como imágenes en la pantalla.

MEMORIA DE TRANSPASO DE ENTRADA DE LINEA

En BASIC, entre $580 y $5FF.

MEMTOP

En BASIC, un puntero ([90,91]decimal) a la parte más alta de la RAM de aplicación, el final del programa de usuario. La expansión del programa puede ocurrir a partir de este punto hacia el final de RAM, que se define por el comienzo de la lista de despliegue. Este MEMTOP no es el mismo que la variable llamada MEMTOP del sistema operativo.

MODO CORTO IRG

En entrada/salida de cassette, ésto se refiere a que la cinta no se detiene entre registros. Los comandos BASIC "CSAVE" y "CLOAD", ambos especifican este modo.

MODO DE CARACTERES

Un tipo específico de los modos de despliegue ANTIC, que interpreta y despliega los datos de la memoria de despliegue como caracteres, recurriendo a un juego de caracteres. Hay 6 modos de caracteres ANTIC, 3 de los cuales son accesibles desde BASIC en los modelos 400 y 800. 5 son accesibles desde BASIC en los modelos XL.

MODO DE DESPLIEGUE

Metodología, ya sea BASIC o ANTIC para interpretar bytes de datos de mapa o texto en la memoria de pantalla y de desplegarlos en la pantalla

MODO DE MAPA

Un modo de despliegue ANTIC específico, que usa pixels de pantalla de colorido simples en vez de caracteres para el despliegue de pantalla. Hay 8 modos de mapa de ANTIC, con diversos grados de resolución. Seis de ellos son accesibles desde BASIC en los modelos 400/800. En los modelos XL hay acceso a dos modos adicionales. A su vez el modo ANTIC $F (BASIC 8) da origen a los modos GTIA 1, 2 y 3 (BASIC 9, 10 y 11).

MODO DE ROTACION

Es una característica provista por el sistema operativo, mediante la cual después de 9 minutos sin haberse tocado una tecla, los colores comienzan a ciclar a través de la pantalla en forma aleatoria y con luminancias reducidas. Esto asegura que, un computador abandonado a su suerte por algunas horas, no queme una imágen estática en la pantalla del televisor.

MODO INMEDIATO

En BASIC el modo en el cual una línea de entrada no está precedida por un número de línea; BASIC de inmediato ejecuta esta línea.

MODO IRG NORMAL

En entrada/salida de cassette es éste un modo en el cual la cinta siempre se detiene después de la lectura de cada registro. Si el computador detiene la cinta y hace su procesamiento con suficiente rapidez, la lectura siguiente puede ocurrir tan pronto, que la grabadora de cassette alcance a percibir solamente una leve caída en la línea de control.

MONITOR

Un programa en ROM que maneja tanto la secuencia de energización como la de reposición del sistema.

NMI

Interrupción no enmascarable (es decir, no puede ser inhibida por el 6502). Tanto la interrupción por lista de despliegue como por borrado vertical son NMI. Pueden ser inhibidos con el registro NMIEM de ANTIC.

NMIEN

El registro de habilitación de interrupción no enmascarable que control la habilitación de las diversas interrupciones, tales como la interrupción de lista de despliegue (DLI).

NUMERO DE SECTOR

Un valor comprendido entre 1 y 719 que designa el sector del diskette al cual el puntero de archivo está apuntando.

OPERADOR

En BASIC, cualquiera de los 46 códigos que en alguna forma mueven o modifican los valores que le siguen.

PAGINA CERO

En el computador ATARI, es el rango de memoria que se extiende entre las ubicaciones $0000 y $00FF.

PARTIDA EN CALIENTE (WARMSTART).

Otro nombre por 'reposición de sistema' (SYSTEM RESET) que es similar a (en sus funciones de inicialización de vectores), pero no idéntico a una partida en frío (coldstart).

PARTIDA EN FRIO

Sinónimo del proceso de energización que realiza una serie de inicializaciones de base de datos del sistema, al conectar la energía al computador. Después de una partida en frío, el sistema traspasa el control al usuario.

PILA DE OPERADORES

En BASIC, una pila de programación donde se ubican los operadores al evaluar una expresión aritmética en BASIC.

PIXEL

La unidad de punto normal de distancia vertical en la pantalla de televisión. El límite normal de un televisor usado con un computador ATARI es de 192 pixels verticalmente.

PMBAS

Un registro que apunta al comienzo del área de jugador-proyecyil.

POKEY

Un circuito digital de entrada/salida que maneja el bus serial de entrada/salida, la generación de audio, el barrido del teclado y la generación de números aleatorios. POKEY tambien digitaliza las entradas anal14gicas de las paletas resistivas y controla los requerimientos de interrupciones enmascarables (IRQ).

PRIMER PLANO

Equivalente a campo de juego o área de la pantalla que se sobrepone directamente al fondo de la pantalla. El primer plano se forma por despliegue de mapa y/o de texto.

PROYECTIL

Una imagen unidimensional en RAM, usada en la gráfica de jugador-proyectil y que tiene 2 bit de ancho. Hay un máximo de 4 proyectiles, uno para cada jugador.

PUNTERO DE ARCHIVO

Para entrada/salida de diskette es un valor que indica la posición actual dentro del archivo, especificando el sector y la cuenta de bytes. El DOS mantiene un puntero de archivo para cada archivo que se encuentre abierto.

REGISTRO

Para entrada/salida de disco un grupo de bytes limitado por EOL ($9B). Para entrada/salida de cassette se trata de un grupo de 132 bytes compuesto por dos caracteres de marca para medida de la velocidad del cassette, Un byte de control, 128 bytes de datos y un byte de suma de cifras.

REGISTRO DE COLOR

Un registro circuital (con su correspondiente ubicación de copia del sistema operativo) que se usa para dar color a varias partes del despliegue de pantalla. Hay 9 registros de color disponibles en el Sistema de Computación Personal ATARI.

REGISTRO DE CONTROL DE PRIORIDAD

Conocido también como PRIOR y con copia en GPRIOR. Este registro especifica qué imagen de campo, jugador o fondo tiene prioridad en el caso de imágenes que se sobrepongan durante el proceso de despliegue de pantalla.

REGISTRO DE POSICION HORIZONTAL

Un registro especializado que contiene un valor definible por el usuario para la posición horizontal del jugador en la gráfica de jugadores-proyectiles. Este valor se da en unidades de compases de color

REGISTRO DE SONIDO

Circuitería que produce audio en el Sistema de Computación Personal ATARI, y que contiene información de frecuencia, volumen y distorsión, pero no de duración del sonido.

REGISTRO VCOUNT

Este registro ANTIC lleva la cuenta de la línea de barrido horizontal que ANTIC está desplegando.

RELOJ DEL SISTEMA

Un reloj provisto por los computadores ATARI, que avanza a la frecuencia de los cuadros de televisión, que en el caso de la norma NTSC es de 59,923334 Hz. La televisión europea (PAL) opera a 50 Hz. Hay 6 relojes de sistema que se sincronizan durante cada proceso de borrado vertical.

RELOJES POKEY

A diferencia de los relojes de sistema, los relojes del circuito POKEY son sincronizados por frecuencias establecidas por el usuario.

RESOLUCION DE DOS LINEAS

Una unidad de resolución vertical de jugadores en la gráfica de jugadores-proyectiles. Cada byte de jugador ocupa dos líneas de barrido horizontales sobre la pantalla y la tabla de cada jugador tiene una longitud de 128 bytes.

RESOLUCION DE UNA LINEA

Unidad de resolución vertical para un jugador en gráfica de jugador-proyectil. Cada byte de jugador ocupa una línea de barrido horizontal en la pantalla y la tabla de cada jugador tiene una longitud de 256 bytes.

ROBO DE CICLO

Ocurre cuando ANTIC interrumpe el procesamiento del 6502 para realizar funciones de acceso directo de memoria (DMA) para propósitos de despliegue de pantalla.

RTCLOK

Uno de los relojes de sistema de 3 bytes de longitud y que se pone al día durante la interrupción de borrado vertical (VBLANK) inmediata. RTCLOK puede usarse como cronómetro en aplicaciones.

RUNSTK

En BASIC, un puntero ([8E,8F]decimal) que apunta al stack de tiempo de ejecución.

SECTOR

En un diskette, este es un área física de 128 bytes. El diskette contiene 40 pistas de 18 sectores por pista.

SENTENCIA

En BASIC, corresponde a una sentencia de códigos que hacen que BASIC realice alguna tarea con significado. En formato LIST, las sentencias se separan por dobles puntos.

SEÑAL PRIMARIA

Contiene la información de luminancia - datos de brillantez, sincronización y, borrado vertical y horizontal - de la señal modulada de televisión.

SETVBV

Una rutina de sistema que, entre otras funciones, pone correctamente los relojes de sistema y dispone las direcciones de los vectores de interrupción definibles por usuario.

SID

Rutina del sistema serial de entrada/salida, que administra la comunicación entre los administradores de dispositivo serial del computador y el bus serial.

SOBREBARRIDO

La expansión de la imagen de televisión por barrido tal que los extremos de la imagen se encuentran fuera de los bordes de la pantalla. Esto garantiza que los bordes de la imagen de televisión no aparezcan desagradables.

STACK DE TIEMPO DE EJECUCION

En BASIC, un stack de software que contiene las direcciones de entrada de regreso de los GOSUB y FOR/NEXT.

STARP

En BASIC, el puntero ([8C,8D]decimal) al área de agrupamientos de strings.

STMCUR

En BASIC, el puntero ([8A,8B]decimal), a la sentencia BASIC actual.

STMATAB

En BASIC, el puntero ([88,89]decimal), a la tabla de sentencias.

TABLA DE NOMBRES DE VARIABLES

En BASIC, la tabla que contiene una lista de todos los nombres de variables que se han ingresado al programa.

TABLA DE SENTENCIAS

En BASIC, es un bloque de datos que incluye todas las líneas de código que han sido ingresadas por el usuario y codificadas por BASIC. En esta tabla también se incluye la línea de modo inmediato.

TABLA DE VALORES DE VARIABLES

En BASIC, una tabla que contiene la información actual de cada variable.

TASA DE BAUD DE ENTRADA

Para entrada/salida de cassette se asume una tasa nominal de 600 baud (bits físicos por segundo); sin embargo esta tasa es ajustada por el SIO para compensar por variaciones de la velocidad del motor, estiramiento de la cinta, etc.

TINTE

El valor de la cuaterna de bits superior del color de un registro de color. Hay 16 posibles tintes ($0 a $F) que en combinación con un valor de luminancia constituye un color definido. Ejemplos de tinte son: negro, rojo, dorado.

TONO DE ESCRITURA PRE-REGISTRO

Una frecuencia de tono puro de marca que se usa como limitador pre-registro en la entrada/salida de cassette.

VARIABLE

En BASIC, un código que es un puntero indirecto hacia su real valor.

VBREAK

Es el vector IRQ de la instrucción BRK del 6502. Cada vez que se ejecuta el código operativo $00 (la instrucción de programa break) ocurre esta interrupción. VBREAK normalmente apunta a una instrucción RTI.

VECTOR DE INTERRUPCION DE LISTA DE DESPLIEGUE (DLI)

Es este un puntero de dos bytes (byte menos significativo, byte más significativo) a la rutina de servicio de interrupción de lista de despliegue. El usuario debe poner este vector, que se encuentra en [512,513] decimal.

VECTOR RAM

Un vector de sistema alterable, que contiene la dirección de dos bytes a alguna rutina de sistema, punteros de entrada de administrador o rutina de inicialización. Los vectores RAM se inicializan durante la energización y SYSTEM RESET.

VECTOR ROM

Un vector de sistema inalterable que contiene instrucciones de salto (JMP) a las rutinas del sistema.

VENTANA DE TEXTO

En un despliegue de pantalla corresponde a un área bi-dimensional separada para entrada/salida con el usuario.

VDSLST

Es el vector de interrupción no enmascarable (NMI) de la lista de despliegue ubicado en [$0200,$0201].

VIMIRQ

Es éste el vector de interrupción IRQ inmediato. Todas las interrupciones IRQ vectorizan a través de esta ubicación. VIMIRQ normalmente apunta al administrador IRQ. Este vector puede modificarse para procesar las interrupciones IRQ del usuario.

VINTER

Es el vector de interrupción IRQ de periférico. Esta línea de interrupción también es accesible a través del bus serial. VINTER normalmente apunta a una instrucción RTI.

VKEYBD

Es el vector IRQ del teclado, que se activa presionando cualquier tecla, excepto BREAK. Este vector normalmente apunta a la rutina IRQ de teclado propia del sistema operativo.

VNTD

En BASIC, el puntero ([84,85]decimal) al extremo ficticio de la tabla de nombres de variables. BASIC usa este puntero para identificar el final de la tabla de nombres. El puntero normalmente apunta a un byte Ø de relleno cuando hay menos de 128 variables. Al haber 128 variables apunta al último byte del último nombre de variable.

VNTP

En BASIC, el puntero ([82,83]decimal) a la tabla de nombres de variables.

VPRCED

Es el vector IRQ de ejecución de periférico. La línea de ejecución está disponible para los periféricos a través del bus serial. Por ahora este IRQ no se emplea y normalmente apunta a una instrucción RTI.

VSCROL

Es el registro de deslizamiento vertical fino ubicado en $D405. El usuario pone en VSCROL el número de línea de barrido en el cual debe deslizarse verticalmente la línea de pantalla.

VSERIN

Es el vector IRQ de entrada serial lista de POKEY.

VSEROR

Es el vector IRQ de salida serial lista de POKEY.

VTIMR1

Es el vector IRQ del reloj 1 POKEY.

VTIMR2

Es el vector IRQ del reloj 2 POKEY.

VTIMR4

Es el vector IRQ del reloj 4 de POKEY.

VVBLKD

Es el vector de interrupción NMI de borrado vertical diferido ubicado en [$0224,$0225].

VVBLKI

Es el vector de interrupción NMI de borrado vertical inmediato ubicado en [$0222,$0223].

VVTP

En BASIC, es el puntero ([86,87]decimal) a la tabla de valores de variables.

WSYNC

Espera del sincronismo horizontal del haz de electrones que está atrasando la imagen de pantalla. El registro WSYNC al ser escrito en cualquier forma, baja la línea RDY del microprocesador 6502, deteniéndolo hasta que el haz de electrones que traza la imagen de pantalla vuelva al extremo izquierdo de la pantalla.

ZIOCB

Bloque de control entrada/salida de página cero - se usa para comunicar datos de control entrada/salida entre el CIO y los administradores de dispositivo.