

ATARI

KOCH

PEEKS Y POKES

PARA

ATARI[®]

600 XL / 800 XL / 130 XE

UN LIBRO EDITADO POR DATA BECKER S.A

Koch / Peeks y Pokes para ATARI 600 XL / 800 XL / 130 XE

KOCH

PEEKs Y POKES

PARA

ATARI[®]

600 XL / 800 XL / 130 XE

UN LIBRO EDITADO POR DATA BECKER S.A

ISBN 84-86437-13-X

Déposito legal B-26.875-85

Copyright (C) 1985

**DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf.**

Copyright (C) 1985

**FERRE MORET S.A.
Tuset n. 8 ent. 2
08006 Barcelona.**

Copyright (C) 1987

**DATA BECKER S.A.
Paraguay 783, 11^º "C"
(1057) - BUENOS AIRES
ARGENTINA.**

Hecho el depósito que marca la ley.

I.S.B.N. 950-99088-2-7

Reservados todos los derechos. Ninguna parte de este libro podrá ser reproducida de algún modo (impresión, fotocopia o cualquier otro procedimiento) o bien, utilizado, reproducido o difundido mediante sistemas electrónicos sin la autorización previa de FERRE MORET S.A.

Impreso en Argentina

Printed in Argentina

Este libro ha sido traducido por Dña. Petra Scheffler
arquitecta técnica y entusiasta del ATARI.

Advertencia Importante

Los circuitos, procedimientos y programas reproducidos en este libro, son divulgados sin tener en cuenta el estado de las patentes. Están destinados exclusivamente al uso amateur o docente, y no pueden ser utilizados para fines comerciales.

Todos los circuitos, datos técnicos y programas de este libro, han sido elaborados o recopilados con el mayor cuidado por el autor y reproducidos utilizando medidas de control eficaces. No obstante, es posible que exista algún error. FERRE MORET, S.A. se ve por tanto obligada a advertirles, que no puede asumir ninguna garantía, ni responsabilidad jurídica, ni cualquier otra responsabilidad sobre las consecuencias atribuibles a datos erróneos. El autor les agradecerá en todo momento la comunicación de posibles fallos.

INDICE

=====

ATARI, BASIC, PEEK Y POKE	1
JUEGOS DE NUMEROS	6
DESFILÉ DE BITS	18
A TRABAJAR CON ROM Y RAM	25
ESQUEMA DE EMPLAZAMIENTO	28
MAPA DE MEMORIA	31
GRAFICO PLAYER-MISSILE	167
SONIDO	190
VARIADO	194
PLANO DE MEMORIA DE PM	196
DISPLAY LIST	198
MEMORIA DE PANTALLA	208
TABLA - MODOS GRAFICOS	223
JUEGO DE CARACTERES	225
LABEL - ADECEDARIO	240
LEXICO DEL ORDENADOR	243

ATARI, BASIC, PEEK Y POKE

El lenguaje de programación BASIC se compone de un conjunto de comandos con los cuales se forman los programas según unas determinadas normas. El ordenador es capaz de ejecutar estas instrucciones gracias al traductor (intérprete) que transforma los comandos BASIC al lenguaje máquina. Dicho de manera más simple, cada comando BASIC llama a una corta rutina en lenguaje máquina que le corresponde.

Sin embargo, a la facilidad de manejo del BASIC se le contrapone la lentitud con que éste ejecuta los programas. Ello es debido, por un lado, al tiempo que se invierte en su traducción y, por otro, a la compleja estructuración de los mismos, mucho más sencilla en un lenguaje directo.

A la mayoría de usuarios no les molesta especialmente esta circunstancia; el hecho de ahorrarse unos microsegundos en la ejecución de los programas apenas tiene importancia para el programador aficionado. Sin embargo, la persona que haya adquirido cierta experiencia en programación BASIC, pronto se dará cuenta de las limitadas posibilidades que permite su léxico.

Pero incluso al llegar a este límite, el lenguaje BASIC no nos deja en la estacada. Utilizando convenientemente los comandos PEEK (vistazo) y POKE (empujón) en un programa BASIC, podemos acceder directamente a las celdas de memoria del ordenador. De esta forma, y en función de la experiencia y conocimientos del programador, se pueden elaborar programas propios haciendo uso del direccionamiento directo.

Para ello es condición necesaria estar al corriente de lo que ocurre en cada uno de los niveles de la memoria del ordenador, de la forma de actuar de los comandos PEEK y POKE, y de las celdas de memoria en las que se pueden efectuar cambios.

Sobre estos particulares es de lo que pretende informarle

este libro.

El corazón del ATARI es un procesador de 8 bits que puede administrar 65536 (0 a 65535) posiciones de memoria (direcciones). Cada una de estas posiciones de memoria se compone de 8 bits (binary digits).

La información generada por un bit corresponde a una decisión si/no, lo que para la máquina significa el paso o no de corriente, es decir, encendido o apagado. Matemáticamente se expresa con 1 o 0.

Un grupo de ocho bits definen un byte. Esta es la menor unidad de la memoria a la que podemos acceder. Cada posición de memoria está ocupada por un byte. En el byte se asigna un lugar de 0 a 7 a cada uno de los bits que lo componen. De esta forma se obtiene un número binario de ocho cifras:

ESTRUCTURA DE UN BYTE:

Número del bit	7	6	5	4	3	2	1	0
Bits (ejemplo)	1	0	0	1	0	1	1	0
Valor binario	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

El número binario de ocho cifras, que a partir de ahora también designaré como bit tipo, puede ser transformado fácilmente en valor decimal. Para ello deben sumarse los valores binarios de todos los bits que se encuentran a "1":

VALOR DECIMAL DE UN BYTE:

Número de bit	7	6	5	4	3	2	1	0
Bits (ejemplo)	1	0	0	1	0	1	1	0
Valor posicional	128	64	32	16	8	4	2	1

En este ejemplo pues, el byte tiene el valor decimal $128+0+0+16+0+4+2+0=150$. Si con el comando POKE n,150 se "empuja" el valor 150 a la dirección n, en ella se definirá el bit tipo del ejemplo. Si con el comando PEEK (n) "se echa un vistazo" a la dirección n donde se encuentra nuestro bit tipo, el comando PEEK encontrará el valor decimal 150.

Así pues en el BASIC del ATARI sólo se trabaja con números decimales. Sin embargo, respecto a los números binarios debe tenerse en cuenta, que la posición de cada bit en un byte corresponde a un valor determinado. El bit n tiene el valor posicional 2 elevado a n . Si además sabemos que 2 elevado a 0 es igual a 1 , podemos empezar a trabajar con PEEK y POKE.

Los valores decimales tienen poca utilidad en la programación en lenguaje máquina; un valor como 150 no indica de entrada los bits activados y desactivados. La forma binaria tampoco es que sea muy práctica con sus largas filas de 0 y 1 . Por ello suelen agruparse a menudo cuatro bits en un solo valor. Cuatro bits pueden representar valores de 0 (0000) a 15 (1111). El número 16 es la base, como su nombre indica, del sistema hexadecimal:

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Ejemplos:

Binario	0000	0010	0101	1000	1010	1111
Decimal	000	002	005	008	010	015
Hexadecimal	0	2	5	8	A	F

Las cifras hexadecimales también pueden agruparse para formar números hexadecimales, lo mismo que las cifras binarias para formar números binarios y las cifras (decimales) para formar números (decimales).

Binario	0000	0000	0001	0010	0100	1101	1111	1111
Decimal		000		018		077		255
Hexadecimal	0	0	1	2	4	D	F	F

Un byte puede tener valores decimales entre 0 y 255 . Estos también pueden representarse en forma hexadecimal con dos cifras (de 00 a FF). Los números hexadecimales tienen la ventaja de ser mucho más manejables que los números binarios, permitiendo además una buena orientación respecto a la estructura matemática del ordenador. Puesto que los

valores binario y hexadecimal están íntimamente ligados, ambos pueden relacionarse con la base, mientras que el sistema decimal no permite prácticamente relación alguna.

Al trabajar con el BASIC del ATARI no hay que preocuparse especialmente por los números hexadecimales. Basta con tener ciertos conocimientos sobre el sistema binario. Sin embargo, dentro de la memoria del ordenador es más fácil orientarse utilizando la notación hexadecimal.

¿Por qué? Siga leyendo.

JUEGOS DE NUMEROS

¡No tan deprisa! Antes de mirar en las profundidades de la memoria, siga manejando un poco más los sistemas de números que se acaban de presentar.

Un sistema numérico es un conjunto de cifras que se combinan para formar números cualesquiera. La posición de una cifra en un número le da su valor.

Habitualmente trabajamos con el sistema decimal, formado por las diez cifras de 0 a 9. Cada posición en un número decimal representa una potencia de diez. La última posición, o posición cero (unidades), tiene el valor 10 elevado a $0 = 1$. La segunda posición (decenas) tiene el valor 10 elevado a $1 = 10$. La tercera posición (centenas) el valor 10 elevado a $2 = 100$. El número 417 se representa pues por $4 \cdot 10$ elevado a $2 + 1 \cdot 10$ elevado a $1 + 7 \cdot 10$ elevado a 0 ó bien $400 + 10 + 7$.

Los demás sistemas numéricos también funcionan de la misma manera. Así pues, el sistema binario sólo utiliza las dos cifras 0 y 1. Cada posición en un número binario corresponde a una potencia de base 2.

El sistema hexadecimal utiliza dieciséis cifras (0 a F). Cada posición en un número hexadecimal corresponde a una potencia de base 16. El número 3E representa el valor $3 \cdot 16$ elevado a $1 + E \cdot 16$ elevado a $0 = 3 \cdot 16 + 14 \cdot 1$, que correspondería al número 62 en notación decimal y al 0011 1110 en binaria (el espacio entre ambos bloques sólo sirve para facilitar la lectura del número y los 0 de las posiciones superiores podrían suprimirse tal como en la notación decimal, donde el número 00531 tiene el mismo valor que el 531).

En informática es habitual separar los ocho bits de un byte en dos bloques. Este bloque de cuatro bits se llama "nibble" y puede comprender valores entre 0 y 15, lo que corresponde

Los matemáticos, de los que es conocida su afición por toda clase de juegos, trabajan con todos los sistemas numéricos imaginables. Así también tiene cierto interés el sistema duodecimal, de base 12, compuesto por las cifras 0 a B. Algunos sistemas de numeración y de medida antiguos utilizan esta base, como por ejemplo, la hora o la pulgada. Tal preferencia se deba seguramente al hecho de que el doce no sólo es divisible entre dos, sino que también lo es entre tres y cuatro.

Puesto que el paso de un sistema numérico a otro es bastante laborioso, sobre todo si se utilizan bases tan diferentes entre sí como 2 y 10 ó 16 y 10, y teniendo en cuenta que el ordenador fue creado precisamente para ahorrarnos tareas de rutina, es lógico confeccionar programas que se encarguen de ello.

El programa DEZXBIN.BEC transforma un número decimal entre 0 y 255 en su correspondiente número binario de ocho cifras. Este programa no está protegido contra errores de entrada:

```

0 REM DEZXBIN.BEC
1 REM *****
2 REM *
3 REM * TRANSFORMACION DE DECIMAL *
4 REM * A BINARIO *
5 REM *
6 REM *****
10 DIM B(7)
20 FOR J=0 TO 7:B(J)=0:NEXT J
30 ? CHR$(125):? "POR FAVOR INTRODUCZA UN NUMERO DECIMAL"
40 ? :? "DE 0 A 255 PULSA <RETURN>":?
50 INPUT D
60 R=D
90 IF D>127 THEN D=D-128:B(7)=1
100 IF D>63 THEN D=D-64:B(6)=1
110 IF D>31 THEN D=D-32:B(5)=1
120 IF D>15 THEN D=D-16:B(4)=1
130 IF D>7 THEN D=D-8:B(3)=1
140 IF D>3 THEN D=D-4:B(2)=1
150 IF D>1 THEN D=D-2:B(1)=1
160 B(0)=D
170 ? :? "DEC ";R;" = "; "BIN
";B(7);B(6);B(5);B(4);B(3);B(2);B(1);B(0)
180 ? :? "OTRAS ENTRADDAS? S/N"
190 IF PEEK(764)=62 THEN POKE 764,255:GOTO 20
200 IF PEEK(764)=35 THEN POKE 764,255:END
210 GOTO 190

```

10: Se da la DIMensión 8 a la variable B (de 0 a 7), para que ésta pueda tomar los ocho bits.

20: CHR\$(125) limpia la pantalla.

90 a 150: Aquí se determinan los bits que están activados. Si el número decimal entrado es mayor que 127 se activa el bit 7 (valor posicional 128). Si se da este caso, se restará 128 del número entrado D. Si éste es ahora mayor que 63, habrá que activar el bit 6 (valor posicional 64), etc.

160: Aquí D ya sólo puede tener valor 0 ó 1 y éste es precisamente el del bit 0 (valor posicional 1).

170: Imprime el resultado.

180: Si el usuario desea transformar otro número decimal, debe entrar "S".

190 y 200: Consultan la entrada realizada en la dirección 764. Más adelante le explicaremos detalladamente cómo funciona esto.

210: La ejecución vuelve hacia 190. De esta forma se crea un bucle del cual sólo puede salirse pulsando la tecla "S" o "N". Otra manera de interrumpir el programa es pulsando BREAK, ya que no está protegido contra ello.

La persona que tenga un poco de experiencia en programación reconocerá rápidamente, que las líneas 90 a 150 tienen el aspecto de poder configurarse de forma más eficiente, por ejemplo, mediante un bucle FOR-NEXT. Sustituya las líneas 90 a 160 del programa anterior por las líneas 110 a 130 del programa DEM0001.bec y compruebe el resultado. Un programa más corto no tiene que ser necesariamente más rápido. El cálculo de la potencia de base dos en la nueva línea 110 requiere más tiempo de lo deseado. La diferencia es sorprendente:

```

0 REM DEM0001.BEC
1 REM *****
2 REM *
3 REM * MAS DESPACIO CON FOR-NEXT *
4 REM *
5 REM *****
110 FOR J=7 TO 1 STEP -1:BW=2^J
120 IF D>BW-1 THEN D=D-BW:B(J)=1
130 NEXT J:B(0)=D

```

El programa siguiente realiza la función inversa. El usuario entra un número binario de ocho cifras y el ordenador halla el correspondiente número decimal.

```

0 REM BINXDEZ.BEC
1 REM *****
2 REM *
3 REM * CONVERSION BINARIO-DECIMAL *
4 REM *
5 REM *****
10 DIM B$(8),A(8)
20
A(8)=1:A(7)=2:A(6)=4:A(5)=8:A(4)=16:A(3)=32:A(2)=64:A(1)=128
30 ? CHR$(125):? "INTRODUZCA UN NUMERO BINARIO"
40 ? :? "CON 8 DIGITOS Y PULSE A CONTINUACION >RETURN<"
50 INPUT B$
60 FOR J=1 TO 8
70 IF B$(J,J)="1" THEN D=D+A(J)
80 NEXT J
100 ? :? "BIN ";B$;"=";"DEC ";D
110 ? :? "DESEA CONTINUAR? S/N"
120 IF PEEK(764)=62 THEN POKE 764,255:D=0:GOTO 30
130 IF PEEK(764)=35 THEN POKE 764,255:END
140 GOTO 120

```

10: Se da la DIMensión 8 a la variable B\$ (de 1 a 8), para que pueda aceptar el número binario. Aunque A tenga la DIMensión nueve (de 0 a 8), sólo se utilizan las variables de A(1) a A(8) para no confundir la relación con B\$. De la misma manera se podría trabajar con A(0) a A(7).

20: Las variables A(8) a A(1) toman los valores decimales que corresponden a los valores posicionales de las respectivas posiciones binarias.

60 a 80: Este bucle FOR-NEXT lee cada uno de los caracteres del número binario de ocho cifras que ha sido entrado anteriormente y almacenado como string en la variable B\$. La Instrucción BASIC del ATARI B\$(n,m) saca un elemento de B\$ comenzando por el carácter que se encuentra en la posición n del string y finalizando con el carácter de la posición m de B\$. De esta forma, B\$(J,J) saca solamente un único carácter de B\$, que es precisamente el que se encuentra en la posición J. Los caracteres de un string se cuentan de izquierda a derecha, o sea, en el sentido de lectura. El primer carácter de B\$ es el bit 7 de valor decimal 128. El valor decimal que corresponde a cada bit ha sido asignado a la variable A(J) en la línea 20.

Al efectuar el paso a números hexadecimales se presenta una dificultad adicional, ya que a las cifras 0 a 9, aceptadas como valores numéricos por el ordenador, se añaden las letras A a F, que el ordenador sólo puede procesar como string. Por ello se suele indicar con \$ el que se trata de un número hexadecimal, o sea, 243 = \$F3:

```

0 REM DEZXHEX,BEC
1 REM *****
2 REM *
3 REM *CONVERS. DECIMAL-HEXADECIMAL*
4 REM *
5 REM *****
10 DIM HEX$(16),H$(1),L$(1)
20 HEX$="0123456789ABCDEF"
30 ? CHR$(125):? "INTRODUZCA UN NUMERO DECIMAL"
40 ? :? "ENTRE 0 Y 255 PULSANDO A CONTINUACION <RETURN>"
50 INPUT D
60 R=D
70 A=INT(D/16):H$=HEX$(A+1,A+1)
80 B=D-A*16:L$=HEX$(B+1,B+1)
170 ? :? "DEC. ";R;"=";" HEX. ";H$;L$
180 ? :? "DESEA CONTINUAR S/N"
190 IF PEEK(764)=62 THEN POKE 764,255:GOTO 30
200 IF PEEK(764)=35 THEN POKE 764,255:END
210 GOTO 190

```

10: HEX\$ recogerá las dieciseis cifras necesarias. El número hexadecimal buscado se compone de dos cifras. La posición superior (HI = high) se recoge en H\$ y la inferior (LO = low) en L\$.

20: Aquí se depositan las cifras de 0 a F en HEX\$.

50: Se almacena en D el número decimal entre 0 y 255 entrado por el usuario.

70: Al dividir D entre 16, la parte entera (INT) del resultado corresponde a la posición superior del número hexadecimal. Pero puesto que este resultado es decimal, todavía se deberá averiguar la cifra hexadecimal correcta de HEX\$: HEX\$(A+1,A+1). Ya que la cifra hexadecimal 0 es el primer carácter en HEX\$, debe sumarse 1 al resultado A.

80: El resto de la división anterior corresponde a la cifra hexadecimal inferior. D-A*16 determina el resto en forma decimal, que será transformado de igual manera en una cifra hexadecimal.

Al pasar de números hexadecimales a decimales se presenta el mismo problema de tener que transformar las cifras

hexadecimales A a F:

```
0 REM HEXXDEZ.BEC
1 REM *****
2 REM *
3 REM * CONVERSION HEXDECIMAL-DECIMAL *
4 REM *
5 REM *****
10 DIM HEX$(2),H$(1),L$(1)
30 ? CHR$(125):? "INTRODUZCA UN NUMERO HEXADECIMAL"
40 ? :? "CON 2 DIGITOS PULSANDO A CONTINUACION <RETURN>"
50 INPUT HEX$
60 H$=HEX$(1,1):L$=HEX$(2,2)
70 IF H$="A" THEN H=10:GOTO 150
80 IF H$="B" THEN H=11:GOTO 150
90 IF H$="C" THEN H=12:GOTO 150
100 IF H$="D" THEN H=13:GOTO 150
110 IF H$="E" THEN H=14:GOTO 150
120 IF H$="F" THEN H=15:GOTO 150
130 H=VAL(H$)
150 IF L$="A" THEN L=10:GOTO 290
160 IF L$="B" THEN L=11:GOTO 290
170 IF L$="C" THEN L=12:GOTO 290
180 IF L$="D" THEN L=13:GOTO 290
190 IF L$="E" THEN L=14:GOTO 290
200 IF L$="F" THEN L=15:GOTO 290
210 L=VAL(L$)
290 D=H*16+L
300 ? :? "HEX.";HEX$;"=";" DEC.";D
310 ? :? " DESEA CONTINUAR? S/N"
320 IF PEEK(764)=62 THEN POKE 764,255:GOTO 30
330 IF PEEK(764)=35 THEN POKE 764,255:END
340 GOTO 320
```

10: El número hexadecimal a entrar sólo puede tener dos cifras. Según esta condición se fija el DIMensionado.

60: Se determinan H\$ y L\$ procedentes del INPUT HEX\$.

70 a 120: Si H\$ es una cifra de A a F, se anota en H el valor decimal de H\$.

130: En caso contrario, se puede transformár la cifra en H\$ directamente en un valor numérico mediante el comando VAL.

150 a 210: Se transforma L\$ de igual manera.

290: Ahora puede calcularse fácilmente el valor decimal buscado. La posición superior del número hexadecimal tiene valor 16 (o sea, $H*16$), la inferior tiene valor 1 (o sea, $1*L$) = $H*16+L$.

Ahora el paso de un número binario a uno hexadecimal tan sólo es una combinación de las rutinas de programa comentadas anteriormente:

```
0 REM BINXHEX.BEC
1 REM *****
2 REM * CONVERSION BINARIO-HEXADECIMAL *
5 REM *****
10 DIM HEX$(16),H$(1),L$(1),B$(8),A(8)
20 HEX$="0123456789ABCDEF"
30 A(8)=1:A(7)=2:A(6)=4:A(5)=8:A(4)=1:A(3)=(2)=4:A(1)=8
40 ? CHR$(125):? "INTRODUZCA UN NUMERO BINARIO"
50 ? :? "CON 8 DIGITOS PULSANDO A CONTINUACION <RETURN>"
60 INPUT B$
70 FOR J=1 TO 4
80 IF B$(J,J)="1" THEN H=H+A(J)
90 NEXT J
100 H$=HEX$(H+1,H+1)
110 FOR J=5 TO 8
120 IF B$(J,J)="1" THEN L=L+A(J)
130 NEXT J
140 L$=HEX$(L+1,L+1)
170 PRINT :PRINT "BIN.";B$;"=";" HEX.";H$;L$
180 ? :? "DESEA CONTINUAR? S/N"
190 IF PEEK(764)=62 THEN POKE 764,255:H=0:L=0:GOTO 30
200 IF PEEK(764)=35 THEN POKE 764,255:END
210 GOTO 190
```

```

0 REM HEXXBIN.BEC
1 REM *****
2 REM *
3 REM **CONVERSION BINARIO-HEXADECIMAL*
4 REM *
5 REM *****
10 DIM HEX$(2),H$(1),L$(1),B(7)
20 FOR J=0 TO 7:B(J)=D:NEXT J
30 PRINT CHR$(125):PRINT "INTRODUZCA UN NUMERO HEXADECIMAL"
40 PRINT :PRINT "CON 2 DIGITOS PULSANDO A CONTINUACION
<RETURN>"
50 INPUT HEX$
60 H$=HEX$(1,1):L$=HEX$(2,2)
70 IF H$="A" THEN H=10:GOTO 150
80 IF H$="B" THEN H=11:GOTO 150
90 IF H$="C" THEN H=12:GOTO 150
100 IF H$="D" THEN H=13:GOTO 150
110 IF H$="E" THEN H=14:GOTO 150
120 IF H$="F" THEN H=15:GOTO 150
130 H=VAL(H$)
150 IF L$="A" THEN L=10:GOTO 220
160 IF L$="B" THEN L=11:GOTO 220
170 IF L$="C" THEN L=12:GOTO 220
180 IF L$="D" THEN L=13:GOTO 220
190 IF L$="E" THEN L=13:GOTO 220
200 IF L$="F" THEN L=15:GOTO 220
210 L=VAL(L$)
220 IF H>7 THEN H=H-8:B(7)=1
230 IF H>3 THEN H=H-4:B(6)=1
240 IF H>1 THEN H=H-2:B(5)=1
250 B(4)=H
260 IF L>7 THEN L=L-8:B(3)=1
270 IF L>3 THEN L=L-4:B(2)=1
280 IF L>1 THEN L=L-2:B(1)=1
290 B(0)=L
300 PRINT :PRINT "HEX.";HEX$;"=";"
BIN.";B(7);B(6);B(5);B(4);B(3);B(2);B(1);B(0)
310 PRINT :PRINT "DESEA CONTINUAR S/N"
320 IF PEEK(764)=62 THEN POKE 764,255:GOTO 20
330 IF PEEK(764)=35 THEN POKE 764,255:END
340 GOTO 320

```

También sobran explicaciones detalladas sobre el paso de números hexadecimales a binarios:

```

0 REM UMWAND.BEC
1 REM *****
2 REM *                               *
3 REM *MENU DE CONVERSION NUMERICA *
4 REM *                               *
5 REM *****
10 DIM HEX$(16),H$(1),L$(1),B$(8),B(7),A(8)
20 HEX$="0123456789ABCDEF"
100 PRINT CHR$(125):POKE 82,0:POKE 752,1
110 ? :? "  TRANSFORMACION DE NUMEROS  "
120 ? "-----"
130 ? :? "  DECIMAL A BINARIO          <1>"
140 ? :? "  DECIMAL A HEXADECIMAL     <2>"
150 ? :? :? :? "  BINARIO A DECIMAL          <3>"
160 ? :? "  BINARIO A HEXADECIMAL     <4>"
170 ? :? :? :? "  HEXADECIMAL A DECIMAL          <5>"
180 ? :? "  HEXADECIMAL A BINARIO     <6>"
200 OPEN# 1,4,0,"K:":GET# 1,T:CLOSE# 1
210 IF T<49 THEN 200
220 IF T>54 THEN 200
230 GOTO (T-48)*1000
1000 REM INICIO PORGRAMA DECIMAL A BINARIO:DEZ IN BIN
2000 REM INICIO PROGRAMA DECIMAL A HEXADECIMAL:DEZ IN HEX
3000 REM INICIO PROGRAMA BINARIO A DECIMAL:BIN IN DEZ
4000 REM INICIO PROGRAMA BINARIO A HEXADECIMAL:BIN IN HEX
5000 REM INICIO PROGRAMA HEXADECIMAL A DECIMAL:HEX IN DEZ
6000 REM INICIO PROGRAMA HEXADECIMAL A BINARIO:HEX IN BIN

```

Si lo desea, Ud. puede resumir estos seis pequeños programas en un completo programa auxiliar (utility). En la pantalla puede aparecer un menú que le permita decidir entre las seis posibilidades de transformación. Pulsando la correspondiente tecla numérica, el programa salta a la parte que se encarga de la operación deseada. El programa UMWAND.BEC muestra la estructura adecuada. Sólo tendrá que añadir los programas precedentes comenzando respectivamente en las líneas 1000 a 6000. Además sería conveniente tomar precauciones contra posibles errores de entrada del usuario (TRAP, TRAP, TRAP):

200: Se abre un canal de lectura hacia el teclado (K=keyboard), se determina el valor ATASCII de la tecla pulsada en la variable T y se vuelve a cerrar el canal de datos.

210 a 220: Estas líneas comprueban si se ha pulsado una de las teclas "1" a "6" (Valores ATASCII 49 a 54).

230: En caso afirmativo, se determina ahora el correspondiente destino de salto.

DESFILE DE BITS

En los ordenadores cada uno de los bit funciona como un interruptor. Ocho de estos interruptores se agrupan en una unidad de control (byte) y se representan en forma de valor decimal (byte de datos). Observando uno de estos valores, por ejemplo 191, es imposible reconocer ni siquiera con mucha práctica cuáles son los bits activados (1) y desactivados (0). No existe tampoco ninguna instrucción BASIC que nos informe directamente sobre el estado de determinados bits (aunque muchas instrucciones en BASIC influyan directamente en cada bit por separado).

Sin embargo, en muchas ocasiones es necesario consultar un byte para determinar el bit tipo almacenado en él, o para comprobar el estado de un bit determinado. El comando PEEK(n) sólo encuentra un valor decimal. Con la instrucción PRINT PEEK(n) se emite en pantalla el byte de datos que se encuentra en la dirección n; con B=PEEK(n) se puede conservar este valor en una variable.

Para conseguir información sobre el bit tipo de una posición de memoria sólo queda la posibilidad de desglosar el byte de datos en sus bits por separado. La manera de efectuar esta operación ya se ha indicado en el programa DEZXBIT.BEC. A continuación se vuelve a mostrar de forma resumida:

```

0 REM BITBREAK.BEC
1 REM *****
2 REM * DESMENUZANDO UN BYTE DE *
3 REM * DATOS *
5 REM *****
10 DIM B(7)
20 X=INT(RND(0)*8)
30 R=PEEK(53770):D=R
40 FOR J=0 TO 7:B(J)=0:NEXT J
50 IF D>127 THEN D=D-128:B(7)=1
60 IF D>63 THEN D=D-64:B(6)=1
70 IF D>31 THEN D=D-32:B(5)=1
80 IF D>15 THEN D=D-16:B(4)=1
90 IF D>7 THEN D=D-8:B(3)=1
100 IF D>3 THEN D=D-2:B(2)=1
110 IF D>1 THEN D=D-2:B(1)=1
120 B(0)=D
130 PRINT :PRINT " EN EL BYTE DE DATOS ";R;" EL BIT#
";X;"="";B(X)
140 GOTO 20

```

20: La casualidad determina cuál de los ocho bits (de 0 a 7) será investigado por el programa.

30: Con ayuda de R, el byte de datos también adquiere un valor aleatorio comprendido entre 0 y 255. En su lugar también podría escribirse $R=INT(RND(0)*256)$.

40 a 120: Se descompone el byte de datos en sus componentes bit.

Esta rutina de programa ocupa relativamente mucha memoria. Si sólo se desea determinar el estado de un bit aislado, se puede resumir la pregunta en una fórmula más reducida, aunque ésta no será procesada con mayor rapidez. Para poder entender esta fórmula, hay que aclarar la relación entre los valores de los diversos bits. El valor del bit $n+1$ es justo el doble que el del bit n . Si se divide el valor de cualquier bit superior por el valor de cualquier bit inferior, el resultado siempre será un número par.

Ejemplo: Sea el byte de datos 128

Número de bit	6	5	4	3	2	1	0
Valor W	64	32	16	8	4	2	1
Cociente (128/W)	2	4	8	16	32	64	128

Si al dividir un byte de datos por el valor de un bit y el resultado es un número par, el bit en cuestión no está activado (0); si el resultado es un número impar, entonces el bit está activado.

Ejemplo: Sea el byte de datos 133

Número de bit	7	6	5	4	3	2	1	0
Valor	128	64	32	16	8	4	2	1
Cociente INT	1	2	4	8	16	33	66	133
Estado del bit	1	0	0	0	0	1	0	1

También de esta forma se puede comprobar, que en el byte de datos 133 se encuentra el bit tipo 10000101. Seguramente se habrá dado cuenta de que al dividir un byte de datos por el valor de un bit no siempre resulta un número entero.

La fórmula que sirve para comprobar si un determinado bit ha sido activado deberá pues realizar los siguientes pasos:

- 1: hallar el valor del bit número X (= 2 elevado a X),
- 2: dividir el byte de datos R por este valor,
- 3: considerar sólo la parte entera del cociente = $\text{INT}(R/2^{\text{elevado a X}})$,
- 4: comprobar si este valor es par o impar.

Un número es par si es divisible por 2, o sea, si el cociente es un número entero. Si se divide un número por dos e $\text{INT}(n/2)=n/2$, entonces n es un número par. Si $\text{INT}(n/2)$ es diferente de $n/2$, entonces n es impar. Finalmente, la fórmula completa sería:

```
0 REM FORMUL01.BEC
10 REM  $\text{INT}(\text{INT}(R/2^X)/2) \neq \text{INT}(R/2^X)/2$ 
```

IF se cumple esta condición, THEN bit X=1 (está activado).

A continuación se muestra un pequeño programa que aprovecha la fórmula antes hallada:

```
0 REM BITPEEK.BEC
1 REM *****
2 REM *
3 REM * DETERMINACION DE UN BIT *
4 REM *
5 REM *****
10 B=0
20 X=INT(RND(0)*8)
30 R=PEEK(53770)
40 IF  $\text{INT}(\text{INT}(R/2^X)/2) \neq \text{INT}(R/2^X)/2$  THEN B=1
50 ? :? "BYTE-DATOS ";R;"ES BIT# ";X;" = ";B
60 GOTO 10
```


20 y 30: Estas líneas generan valores aleatorios para la posición X del bit y para el byte de datos R.

40: La extraordinaria fórmula entra en acción.

50: Se emite el resultado.

Naturalmente se podría colocar un bucle FOR-NEXT en las líneas 40 y 50, que repasaría los ocho números de bits (de 0 a 7). En este caso habría que dar a B la DIMensión 8 (DIM B(7)). Aún así, el programa sería considerablemente más corto que el BITBREAK.BEC, ocupando por lo tanto menos memoria. Sin embargo, su ejecución sería bastante más lenta, con lo cual esta fórmula sólo adquiere interés cuando se comprueba el estado de un bit por separado.

Para trabajar con el BASIC del ATARI no hace falta saber calcular con números binarios. Los PEEKs y POKEs se realizan con valores decimales. Si se desea modificar el valor en una dirección, hay que determinar el valor actual de la celda n con PEEK(n), cambiarlo sumando o restándole un valor decimal y volver a colocar el nuevo valor en la posición de memoria n. Resumiendo: POKE n, PEEK(n+/-m). Si n-m es menor que cero o n+m es mayor que 255, entonces se emite el mensaje de error ERROR- 3 (fuera del campo numérico).

Para finalizar, un detalle de utilidad. En algunas ocasiones se requiere la conversión de un bit tipo, o sea, escribir un 0 allí donde exista un 1 y al revés. Este proceso se denomina invertir. Se utiliza frecuentemente en aplicaciones gráficas.

Bit tipo inicial	1	0	0	1	0	1	1	0	=	150
(valor decimal)										
+ bit tipo invertido	0	1	1	0	1	0	0	1	=	105
(valor decimal)										
=	sumas	1	1	1	1	1	1	1	=	255
(valor decimal)										

El proceso inverso funciona por lo tanto:

Los ocho bits activados (valor decimal)	1 1 1 1 1 1 1 1	= 255
- bit tipo (valor decimal)	1 0 0 1 0 1 1 0	= 150
= bit tipo inverso (valor decimal)	0 1 1 0 1 0 0 1	= 105

Para invertir un bit tipo sólo hay que restar su valor decimal de 255 y con ello se obtiene el valor decimal del bit tipo inverso. `POKE n,255-PEEK(n)` invierte el bit tipo en la dirección n.

A TRABAJAR CON ROM Y RAM

La constitución del microprocesador determina el número de posiciones de memoria que pueden administrarse. El corazón del ATARI es un 6502, que puede acceder a 65536 celdas. Para representar valores de 0 a 65536 se necesitan ocho bits = dos bytes. Cada 256 bytes se agrupan en una página (page). El byte con valor superior (HI) indica la página de memoria, lo que significa que pueden administrarse 256 páginas. El byte con valor inferior (LO) indica la posición de memoria en la página. De esta manera pueden clasificarse $256*256=65536$ celdas de memoria con un número llamado dirección.

Para registrar una determinada dirección se necesitan pues dos bytes, que siempre se escriben en el orden LO, HI. La dirección buscada resultará entonces de la fórmula $LO + HI*256$. Por ejemplo, si en las direcciones 88 y 89 se ha almacenado un puntero (vector) hacia cualquier dirección, la siguiente instrucción encontrará la dirección buscada:

```
DIRECCION = PEEK(88)+PEEK(89)*256
```

Además de los vectores de dos bytes también existen punteros que sólo ocupan un byte, pudiendo señalar pues solamente el principio de una página. En este caso la dirección buscada se hallaría haciendo:

```
DIRECCION = PEEK(106)*256
```

4 páginas de 256 bytes cada una, o sea 1024 bytes, se designan con el nombre de kbyte (kilobyte). En este caso, la unidad de medida kilo se utiliza modificando ligeramente su dimensión normal. En algunos casos hay que tener en cuenta además los límites de 4 k. Si se accede a la memoria en

forma hexadecimal, o sea de \$0000 a \$FFFF, entonces la mayor cifra de este número hexadecimal cuenta siempre 4 kbyte.

El ordenador dispone de una memoria de 64 kbytes en total (que son 65536 bytes). Sin embargo, el usuario no dispone de la totalidad de esta RAM. Se necesitan aproximadamente la mitad de todas las direcciones para el funcionamiento del ordenador y de los diversos periféricos, así como para almacenar determinados datos que siempre tienen que estar disponibles.

Estos datos están parcialmente anclados en la estructura de la memoria, de manera que se conserven incluso al cortar el suministro eléctrico. Naturalmente el usuario no puede modificar estos contenidos de memoria. Esta parte de la memoria se llama ROM (read only memory = memoria de lectura).

La posiciones de memoria que pueden tomar valores variables se designan con el nombre de RAM (random access memory = memoria con acceso al azar o memoria de lectura/escritura). El sistema utiliza un gran número de estas direcciones RAM para memorizar valores variables como, por ejemplo, punteros determinados. Muchos de estos registros pueden ser manipulados por el usuario, introduciendo determinados valores con ayuda de POKE. Sin embargo, el sistema renueva algunos de ellos tan rápidamente, que esta intervención con el POKE del BASIC no tiene ningún efecto.

En algunos registros cada bit tiene una función determinada, mientras que en otros, ciertas tareas pueden ser realizadas por agrupaciones de varios bits. No siempre se aprovechan los ocho bits disponibles en una celda de memoria.

Al conectar el ATARI (800/XL), el usuario dispone para sus programas y datos de unos 37 kbyte, 148 páginas o mejor dicho de 37902 bytes. Si se conecta una unidad de diskettes, estos valores se reducen a unos 31,5 kbytes, 126 páginas o 32274 bytes, puesto que el DOS también ocupa una parte de la memoria. El comando en lenguaje BASIC PRINT FRE(0) da

información sobre la memoria disponible.

ESQUEMA DE EMPLAZAMIENTO

d=\$-dirección Función

0 = \$0000	Dirección inferior (bottom of memory)
0 = \$0000	OS página cero RAM
128 = \$0080	BASIC página cero RAM
256 = \$0100	6502 stack
512 = \$0200	Puntero interrupción, paddles, registros de color, etc.
768 = \$0300	Parámetros unidad de disco, IOCB 0 - 7, buffer de impresora
1024 = \$0400	buffer del cassette
1152 = \$0480	OS-RAM
1536 = \$0600	libre para rutinas cortas en lenguaje máquina
1792 = \$0700	BOOT-RAM para aplicaciones o DOS, si cargado
2048 = \$0800	RAM o DOS si está cargado
11008 = \$2B00	RAM (el final de DOS fluye)
32768 = \$8000	RAM o módulo ROM derecho (≠)

List Display, memoria de pantalla, la dirección de inicio depende del modo gráfico elegido. Dirección final incluyendo ventana de texto siempre es 40959 = \$9FFF

40960 = \$A000	RAM o módulo ROM izq. (≠) / ROM BASIC (XL)
49152 = \$C000	ROM disponible (≠), OS-ROM (XL)
53248 = \$D000	CTIA (US), GTIA (D)
53504 = \$D100	libre para ocupaciones posteriores
53760 = \$D200	POKEY
54016 = \$D300	PIA
54272 = \$D400	ANTIC
54528 = \$D500	Control de módulo interfase
54784 = \$D600	Libre para ocupaciones posteriores
55296 = \$D800	Chips I/O, floating-point ROM
57344 = \$E000	juego de caracteres estándar ROM
	aquí comienza el OS-ROM, el sistema operativo. Llega hasta 65535 = \$FFFF
58368 = \$E400	Editor
58384 = \$E410	Vectores de pantalla (screen)
58400 = \$E420	Vectores de teclado (keyboard)
58416 = \$E430	Vectores de impresora (printer)
58432 = \$E440	Vectores de cassette de programas (cassette)
58448 = \$E450	Direcciones de salto (vectores JMP)
58496 = \$E480	Vectores RAM para powerup

58534 = \$E483 CIO
59093 = \$E605 Motor Interrupción
59716 = \$EC00 SIO
60906 = \$EDEC Motor unidad de disco
61048 = \$EE78 Motor Impresora
61249 = \$EF41 Motor cassette
61667 = \$F0E3 Rutina monitor
62436 = \$F3E4 Motor pantalla y teclado
65535 = \$FFFF Dirección máxima (top of memory)

MAPA DE MEMORIA

0,1 \$0,\$1 LINZBS

Utilizada por la rutina reset al comprobar la memoria. Es renovada por la RAM del monitor y almacena posiblemente el timer del VBLANK.

2,3 \$2,\$3 CASINI

Vector para inicializar el boot del cassette. Si este proceso da resultado se produce un salto (JSR) hacia la dirección indicada. Los bytes quinto (LO) y sexto (HI) de un fichero de cassette contienen la dirección de inicialización.

4,5 \$4,\$5 RAMLO

Vector RAM para comprobar el tamaño de memoria en el powerup. También se utiliza para la dirección de boot de la unidad de disco (1798 = \$706).

6 \$6 TRAMSZ

Registro intermedio para comprobar el tamaño de memoria. Es utilizado durante el powerup y después transfiere su valor al RAMTOP (106 = \$6A). Si en el conector para cartuchos (XL), o sea el conector izquierdo, se encuentra un módulo de expansión, por ejemplo BASIC, (≠), se leerá un 1.

7

\$7

TSTDAT

Registro de datos para comprobar la RAM. Toma el valor de la posición que se está comprobando. Si se lee un 1 significa que en el conector derecho se encuentra un cartucho (≠).

8

\$8

WARMST

Identificación de arranque en caliente (flag). En este lugar el powerup inicializa un 0. Este valor se mantiene hasta que sea modificado mediante un POKE o hasta pulsar RESET. Este último coloca aquí un 255 (todos los bits activados).

9

\$9

BOOT?

Un boot satisfactorio de la unidad de disco coloca en este lugar un 1 (bit 0 activado), y un boot de cassette un 2 (bit 1 activado). BOOT? contiene un 0 cuando en ninguno de ambos periféricos se haya efectuado un boot. Indica si con RESET se utiliza el vector del cassette (CASINI 2,3 = \$2,\$3) o el del DOS (DOSVEC 10,11 = \$A,\$B). El arranque en frío intenta ambos booting y activa entonces el bit correspondiente.

Si en BOOT? se coloca un 255 (todos los bits activados) se suspende el sistema al pulsar RESET. Esto significa que después de efectuar el boot se puede bloquear la tecla RESET (protección de software).

10,11

\$A,\$B

DOSVEC

Vector de inicio para software de diskette. En BASIC el comando en DOS provoca un salto hacia esta dirección. El puntero puede cambiarse, pero será renovado con RESET. Para evitarlo deben modificarse las direcciones 5446 (LO) y 5450 (HI) = \$1546,\$154A, que habitualmente señalan hacia DOSVEC.

El siguiente programa le mostrará la forma de sustituir el comando DOS con ayuda de DOSVEC:

```
1 REM *****
2 REM *
3 REM *EL DOS A TRAVES DE PULSACION DE TECLA*
4 REM *
5 REM *****
10 REM PROGRAMA
20 REM PROGRAMA
30 REM PROGRAMA
40 REM PROGRAMA
50 REM PROGRAMA
60 IF PEEK(764)=225 THEN 10
70 OPEN# 1,4,0,"K:":GET# 1,D:CLOSE# 1
80 IF D=68 THEN 100
90 GOTO 10
100 LET DOS=USR(PEEK(10)+PEEK(11)*256)
```

10 a 50: simbolizan cualquier programa.

60: Esta línea comprueba si se ha pulsado una tecla del keyboard.

70: En caso afirmativo, se asignará a la variable D el valor ATASCII de la tecla pulsada a través de un canal de datos hacia el teclado.

80: Si el valor de D es 68, significa que se ha pulsado la tecla "D" y el programa se bifurca hacia 100.

90: Salto que retrocede a la línea 10 en caso de no hallar la tecla "D".

100: ATARI incluso funciona si se utiliza una palabra BASIC (en este caso DOS) como variable. Sólo hay que indicarlo expresamente con LET.

El comando USR salta hacia una rutina en lenguaje máquina que comienza por la celda de memoria indicada detrás de USR. Si ahora se utiliza DOSVEC como dirección inicial, entonces USR salta hacia el DOS. El comando USR debe asignarse a una variable (en este caso DOS) que no tenga ninguna otra función.

12,13

\$C,\$D

DOSINI

Vector de inicialización para el boot de la unidad de disco. Toma la dirección donde comienza el programa de aplicaciones después de cargar el DOS. Con un JSR indirecto hacia esta dirección puede inicializarse el software de aplicaciones.

14,15

\$E,\$F

APPMHI

Vector en la primera posición de memoria libre en la RAM de aplicaciones. Indica el límite de la memoria ocupada por un programa en BASIC. La memoria de pantalla y el display list pueden extenderse desde la dirección señalada por este vector hasta RAMTOP (40959 = \$9FFF).

16

\$10

POKMSK

Máscara de interrupción para POKEY. Registro de sombra de IRQEN (53774 = \$D20E). Cada uno de los ocho bits se refiere a una posible interrupción. Al no activar un bit se imposibilita la correspondiente interrupción:

Bit 7	(128)	Tecla BREAK
Bit 6	(64)	Otra tecla
Bit 5	(32)	Datos de input en serie preparados
Bit 4	(16)	Datos de output en serie requeridos
Bit 3	(8)	Salida en serie finalizada
Bit 2	(4)	Timer de POKEY 4
Bit 1	(2)	Timer de POKEY 2
Bit 0	(1)	Timer de POKEY 1

¡Con esto se pueden mantener alejados los usuarios curiosos! Aquí y en IRQEN (53774 = \$D20E) hay que colocar valores con POKE. Cualquier valor inferior a 128 bloquea la tecla BREAK. Aunque con 127 se detenga el programa al pulsar BREAK, a la vez se suspende el sistema. Entonces RESET será la única solución. Pero ya se explicó en BOOT? la manera de proteger la tecla RESET contra cualquier uso indebido. Haga algunas pruebas con valores diferentes.

Sin embargo, cualquier comando dirigido a la pantalla (screen "S:" o editor "E:") coloca un 1 en el bit 7, liberando así la interrupción para BREAK. Después de cada comando PRINT, GRAPHICS u OPEN ("S:" o "E:") hay que

bloquear de nuevo la tecla BREAK. La mejor manera de hacerlo es a través de esta corta subrutina:

```
0 REM ADR168BRK.BEC
1 REM *****
2 REM *
3 REM * SUBROUTINA BREAK-BLOCAJE *
4 REM *
5 REM *****
10 GOSUB 5000
5000 BRK=PEEK(16)
5010 IF BRK>127 THEN BRK=BRK-128
5020 POKE 16,BRK
5030 POKE 53774,BRK
5040 RETURN
```

Las ROM de las más recientes OS versiones "B" tienen un vector aparte para la interrupción BREAK. Referente a ello vea BRKKEY (566,567 = \$236,\$237).

Flag de la tecla BREAK. En esta posición aparece un 0 si se pulsa BREAK. Cualquier otro valor indica que BREAK no está siendo pulsado.

18,19,20

\$12,\$13,\$14

RTCLOK

Este es el reloj interno del ATARI. Funciona en relación con la frecuencia del suministro eléctrico. Mientras que la frecuencia de los aparatos norteamericanos es de 60 hertz, los de aquí trabajan con 50 hertz. Al trabajar con software procedente de los EE.UU. que lea el reloj interno, se deben cambiar los valores con el factor 5/6 para adaptarlo a nuestro ritmo de trabajo más tranquilo.

Al conectar el ordenador, el reloj interno empieza a contar desde cero. El registro 20 cuenta al compás de 50 hertz de 0 a 255, o sea en un segundo de 0 a 49. Al llegar al valor 255, el registro 20 incrementa en 1 el valor del registro 19 y vuelve a empezar por 0. Cuando el registro 19 pase de 255 a 0, aumenta por su parte el valor del registro 18 en 1.

De esta manera, estos tres registros juntos pueden contar hasta 16.777.215. El registro 20 tarda unos 335.544,32 segundos en cambiar de valor, o sea 5.592 minutos o unas 93 horas, que son casi 4 días, o exactamente 3 días 21h 12min. 24s y 32 décimas de segundo. A continuación se muestra un programa para el reloj digital marca ATARI:


```

0 REM ADR18.BEC
1 REM *****
2 REM *
3 REM *RELOJ INTERNO DE TIEMPO REAL*
4 REM *
5 REM *****
10 FOR J=0 TO 2:POKE 18+J,0:NEXT J
20 PRINT CHR$(125):POKE 752,1
30 T=INT(PEEK(18)*65536+PEEK(19)*256+PEEK(20))/(50)
40 D=INT(T/86400)
50 HH=INT(T/3600):H=HH-D*24
60 MM=INT(T/60):M=MM-D*1440-H*60
70 S=T-D*86400-H*3600-M*60
80 POSITION 10,11
90 PRINT D;" d : ";H;" h : ";M;" : ";S;CHR$(34)
100 GOTO 30

```

10: Al iniciar el programa el reloj debe ponerse en primer lugar a 0, puesto que está contando desde el momento en que se conecta el ordenador.

20: Limpia la pantalla y suprime el cursor.

30: Calcula el actual valor numérico de los registros 18 a 20, lo divide por 50 (hertz = vibraciones/segundo) y determina la parte entera del cociente en T.

40: Averigua a cuántos días corresponde el valor de T y asigna el resultado a la variable D.

50: De igual manera determina la cantidad de minutos enteros de T, y HH recoge el resultado. Al restar de HH las horas de los días antes hallados (= D*24), sólo quedan las

horas en curso, valor que se asigna a H.

60: De manera análoga se calculan también los minutos en curso.

70: Los segundos en curso S resultan de restar a la suma total de segundos T los equivalentes en segundos de D, H y M.

80 y 90: El resultado se muestra en la pantalla con PRINT y con

100: se vuelve a ejecutar el programa.

En vez de calcular días, horas, minutos y segundos y esperar a que los registros se llenen y cambien de valor, también se puede hacer que este reloj sólo cuente 24 horas, volver a graduar entonces RTCLK a 0 y contar con ello un día en una variable. De esta forma el reloj puede contar indefinidamente. Se llega a las 24 horas cuando RTCLK tiene el valor $24*60*60*50 = 4.320.000 = 65*65536 + 235*256 + 0$. La instrucción será pues:

```
IF PEEK(20)=65 THEN IF PEEK(19)=235 THEN IF PEEK(18)=0 THEN  
POKE 18,0:POKE 19,0:
```

Naturalmente, un programa que lee el reloj interno también requiere cierto tiempo de ejecución, durante el cual el reloj interno no para de contar. Pero para realizar un programa tan corto, incluso el BASIC es lo suficientemente rápido como para mostrar cada segundo en la pantalla, aunque los registros 18 a 20 sólo serán leídos cuando el programa procesa la línea 30.

Realmente un segundo es un tiempo muy largo para el ATARI, que en este período realiza unos 40.000 ciclos de máquina. Así, mientras que el registro 20 cuente 1 más, determinando con ello 1/50 segundos, el ordenador ha efectuado unos 8.000 tiempos. Incluso si utiliza este reloj digital para formar

parte de un programa largo, éste tendrá que ser o muy extenso o contener cálculos muy complejos (funciones angulares, raíces, potencias, etc.) para que los segundos se emitan en pantalla en pasos dobles.

No es absolutamente necesario transformar el reloj interno en tiempo real. Usted puede consultar por ejemplo sólo el registro 19, que cuenta aproximadamente décimas de segundo, y convertir el valor encontrado en una barra gráfica que aumente o disminuya lentamente. También puede fijar un valor que una vez alcanzado efectúe una orden determinada.

```
IF PEEK(18)*65536+PEEK(19)*256+PEEK(20) THEN IF  
PEEK(18)*65536+PEEK(19)*256+PEEK(20)=X THEN ...
```

21,22 \$15,\$16 BUFADR

Puntero momentáneo del SIO en operaciones de diskette.
Registro para la indirecta dirección del buffer.

23 \$17 ICCOMT

Comando vector para CIO. Se utiliza para encontrar el offset en la tabla de comandos.

24,25 \$18,\$19 DSKFMS

Vector para el FMS.

26,27 \$1A,\$1B DSKUTL

Vector para DUP.

28 \$1C PTIMOT

Desconexión por límite de tiempo de la impresora (timeout).

29 \$1D PBPNT

Puntero del buffer de impresora. Señala hacia la posición actual (byte) en el buffer de impresora. Valores desde 0 hasta valores resultantes de PBUFSZ.

30 \$1E PBUFSZ

Tamaño del buffer de impresora en el modo en cuestión. El tamaño del buffer normal es de 40 bytes.

31 \$1F PTEMP

Utilizado por el motor de la impresora para memorizar provisionalmente el valor del carácter que deberá ser enviado a la impresora.

32 \$20 ICHIDZ

Número índice del motor, que el OS colocará en la tabla de los nombres de ficheros para indicar un fichero abierto. Si no hay ningún fichero abierto, aquí aparece un 255.

33 \$21 ICDNOZ

Indica el número máximo de unidades de diskettes. Es inicializado en 1.

34 \$22 ICCOMZ

Un byte de instrucción que debe ser fijado por el usuario. Determina la manera de formatear los IOCB restantes y qué proceso I/O deberá ser ejecutado.

35 \$23 ICSTAZ

Byte de estado referente al último periférico tratado.

36,37 \$24,\$25 ICBALZ/HZ

Dirección de buffer para la transmisión de datos o dirección del nombre de fichero para comandos como STATUS, OPEN, etc.

38,39 \$26,\$27 ICPTLZ/HZ

Dirección de la rutina put-byte. Una instrucción CLOSE sitúa este puntero hacia "IOCB not OPEN" del CIO.

40,41 \$28,\$29 ICBL LZ/HZ

Contador para la longitud del buffer en PUT y GET. Se reduce en 1 por cada byte transmitido.

42 \$2A ICAX1Z

Primer byte de la información auxiliar que se utiliza con OPEN para concretar la forma del acceso necesario a datos.

43 \$2B ICAX2Z

Segundo bit de las informaciones auxiliares, variables de trabajo CIO.

44,45 \$2C,\$2D ICAX3Z/4Z

Se utiliza en BASIC en las Instrucciones NOTE y POINT para transferir los números de sector.

46 \$2E ICAX5Z

Determina el byte del sector fijado por ICAX3Z/4Z al cual debe accederse.

47 \$2F ICAX6Z

Byte libre. Se utiliza como almacén provisional para el byte de carácter en una operación PUT que se está ejecutando.

48 \$30 STATUS

Memoria interna de estado. Las rutinas SIO utilizan este byte para colocar el estado de la operación SIO que se está ejecutando.

49 \$31 CHKSUM

Suma de comprobación, utilizada por SIO.

50,51 \$32,\$33 BUFRLO/HI

Puntero hacia el buffer de datos. Señala hacia el byte que debe ser emitido o recibido.

. ,53 \$34,\$35 BFENLO/HI

Puntero hacia el siguiente byte detrás del final del buffer de datos.

54 \$36 CRETRY

Número de repeticiones de una instrucción realizadas por un dispositivo después de un intento fallido, como por ejemplo, al formatear el diskette o al escribir en un sector. El valor default es 13.

55 \$37 DRETRY

Número de repeticiones al dirigirse a un dispositivo (por ejemplo, unidad de disco). El valor default es 1.

56 \$38 BUFRFL

Identificación del buffer de datos. 255 significa que está lleno.

57 \$39 RECVDN

Identificación de la recepción de datos. 255 significa que se ha realizado.

58 \$3A XMTDON

Identificación de la transferencia de datos. 255 significa que se ha realizado.

59 \$3B CHKSNT

Identificación de las sumas de comprobación. 255 significa emitido.

60 \$3C NOCKSM

No identificación de sumas de comprobación. 0 significa, que a la transferencia de datos le sucede una suma de comprobación; cualquier otro valor indica que no sigue ninguna suma de comprobación.

61 \$3D BPTR

Puntero del buffer del cassette, indica los datos del registro que en este momento deben ser escritos o leídos.

62 \$3E FTYPE

Determina la distancia entre los bloques de datos.

63 \$3F FEOF

Identificación del motor de cassette para EOF.

64 \$40 FREQ

En este registro, el motor de cassette coloca la cantidad de beeps que deberá emitir el altavoz del ordenador: una vez para la repetición, dos veces para la toma.

65 \$41 SOUNDR

Identificación para la señal de sonido al transferir datos.

Un 0 colocado en este registro mediante POKE provoca que la transferencia sea silenciosa, mientras que cualquier otro valor permite el feedback acústico. Puede utilizarse, por ejemplo, para silenciar la pista digital en las cintas con pistas síncronas de voz y datos.

66 \$42 CRITIC

Un valor diferente de 0 provoca que una serie de procesos OS se suspendan. Esto puede ser necesario cuando se espera una frecuencia de interrupciones alta. Un efecto secundario consiste en que la función repetitiva de teclas se interrumpa, lo que impide pulsar dos veces consecutivas la misma tecla. También se modifica el sonido del zumbador (CONTROL y 2). Sin embargo, es aconsejable no activar CRITIC con un valor diferente de 0.

67-73 \$43-\$49 FMZSPG

Registro de la página cero del FMS. Siete bytes.

74 \$4A CKEY

Identificación de lectura del boot del cassette en arranque en frío. Comprueba si se mantiene pulsado START durante la conexión (arranque en frío). En caso afirmativo, se activa CKEY. Con ello se posibilita la acción automática del boot de cassettes.

75 \$4B CASSBT

Identificación para el boot del cassette. Durante el powerup el OS intenta realizar el boot en cassette y diskette al mismo tiempo. De no haber sido posible un boot de cassette, aquí aparece un 0. Vea BOOT? (9 = \$9).

Registro para el estado del indicador y del teclado. Es utilizado por el motor del indicador. Aquí también se indica, por ejemplo, una posición errónea del cursor, la falta de memoria para representar un modo gráfico o el estado de interrupción mediante BREAK.

Cuando una imagen permanece demasiado tiempo en pantalla, puede quemar la capa fosforescente del tubo del televisor. Parece que los fabricantes de ATARI ya están acostumbrados a los frecuentes descuidos por parte de sus clientes, puesto que han implementado el modo attract. Cuando pasa cierto período de tiempo sin que se efectúe ninguna entrada a través del teclado, este modo varía en intervalos cortos los valores de los registros de color y reduce la claridad general de la pantalla.

Esta disposición sólo tiene el inconveniente de ser molesta. Otros ordenadores personales funcionan sin este tipo de protector de pantalla, y no por ello se ha sabido del quemado de numerosos tubos de pantalla. El modo attract es especialmente molesto porque no se interrumpe con otro tipo de entradas como, por ejemplo, el joystick.

ATTRACT actúa como identificación y timer para el modo attract. El IRQ activa el registro a 0 cada vez que se pulsa una tecla del keyboard. Las teclas de función SELECT, OPTION Y_START carecen aquí de efecto. Sin embargo, la pulsación de una tecla que normalmente no afecta al programa escribe un 0, aunque el programa que se esté ejecutando ni tan siquiera consulte el teclado.

indexado por el VBLANK. Cuando ATTRACT supere el valor 127 el timer sobrepasa su propia capacidad (del bit 0 a 6) y activa el flag (bit 7): el valor decimal del registro cambia a 254 y se activa el modo attract.

El siguiente programa de demostración le muestra la forma de trabajar de este registro. Al mismo tiempo utiliza el reloj interno para cronometrar el tiempo transcurrido hasta activarse el modo attract:

```

0 REM ADR77.BEC
1 REM *****
2 REM *
3 REM * TRATAMIENTO DE PANTALLA *
4 REM * PRG-DEMO *
5 REM *
6 REM *****
10 DIM B(6)
20 POKE 19,0:POKE 20,0
30 ? CHR$(125):POKE 752,1
40 T=INT((PEEK(19)*256+PEEK(20))/50)
50 X=PEEK(77)
60 IF X>127 THEN R=10:POSITION 3,6:? "STOPPED"
70 FOR J=0 TO 6:B(J)=0:NEXT J
80 M=INT(T/60)
90 S=T-M*60
100 Y=X
110 IF X>127 THEN X=X-128:B(6)=1
120 IF X>63 THEN X=X-64:B(5)=1
130 IF X>31 THEN X=X-32:B(4)=1
140 IF X>15 THEN X=X-16:B(3)=1
150 IF X>7 THEN X=X-8:B(2)=1
160 IF X>3 THEN X=X-4:B(1)=1
170 IF X>1 THEN X=X-2:B(0)=1
200 POSITION 2,2:?
CHR$(17);CHR$(18);CHR$(18);CHR$(18);CHR$(18);CHR$(18);CHR$(18)
);CHR$(18);CHR$(5)
210 POSITION 2,3:? CHR$(124);" ";CHR$(124)
220 POSITION 4,3:? M;" "
230 POSITION 6,3:? ":"
240 POSITION 7,3:? S;" "
250 POSITION 2,4:?
CHR$(26);CHR$(18);CHR$(18);CHR$(18);CHR$(18);CHR$(18);CHR$(18)
);CHR$(18);CHR$(3)
260 POSITION 0,16:? "-----"
270 POSITION 3,20
280 ? "AATTRACT:0 ";B(6);" ";B(5);" ";B(4);" ";B(3);"
";B(2);" ";B(1);" ";B(0);" ";X;" = "
290 POSITION 34,20:? Y;" "
300 GOTO 40+R

```

10: B\$ debe recoger los bits de ATRACT. El string se DIMensiona sólo en 6 para ahorrar memoria. Más adelante verá la razón por la que esto es suficiente.

20: Para esta utilidad bastan los registros 19 y 20 de RTCLOCK. Juntos funcionan casi 20 minutos. Aquí son activados a 0.

40: Se calcula el tiempo total T en segundos según el método habitual.

50: En X se anota el valor del registro 77.

60: Cuando X sobrepasa el valor crítico de 127 se para el reloj interno. Esto ocurre al activar la variable R a 10, de manera que el salto hacia atrás al final del programa ya no se dirija a la línea 40, donde se lee RTCLOCK, sino a la 50. Al mismo tiempo debajo del cronómetro aparece la indicación "STOPPED".

70: Los seis elementos de la variable indexada B son activados a 0.

80 y 90: Calculan el tiempo en minutos y segundos.

100: El valor de X se deposita en Y,

110 a 170: después se determinan los bits 7 a 1 del byte de datos X y se anotan en B(J). Puesto que siempre debe restarse de X el valor correspondiente, después de la línea 170 el valor de X equivale al estado del bit 0. Esto explica la línea 10. Cabe mencionar que no es necesario volver a poner a 0 la variable X después de cada pasada, ya que X se llena cada vez con el valor del registro 77.

200 a 260: Imprimen los gráficos de pantalla. Puesto que una impresora de aguja es incapaz de convertir los pseudo-caracteres gráficos sin ayuda de la consiguiente adaptación software, se ha utilizado el comando CHR\$ para los elementos gráficos en cuestión.

220 a 240: Imprimen los minutos y segundos en curso en el recuadro señalado para tal fin.

300: Bucles sin fin mediante saltos hacia atrás.

Si se manipula el teclado durante la ejecución del programa en curso, aunque el valor del registro 77 quede postergado, el cronómetro continúa funcionando. Mediante la lectura de CH (764 = \$2FC) se podría registrar la pulsación de una tecla, y con un salto hacia la línea 20 se podría volver a poner a 0 el cronómetro.

Con este programa, usted mismo podrá darse cuenta del tiempo que requiere ATTRACT para activar el modo attract. El valor es bastante superior al que se asegura en muchas publicaciones. Si escribe un juego o programa gráfico que sobrepase este límite temporal sin que se efectúe ninguna entrada a través del teclado, porque está jugando por ejemplo con el joystick, entonces debería evitar que el modo attract haga "bailar" los colores. Para ello deberá añadir un POKE 77,0 en su programa.

78 \$4E DRKMSK

Aquí aparece un 254 mientras el modo attract no esté activado y después se pone a 246, lo que evita sobrepasar la claridad del color en un 50%.

79 \$4F COLRSH

Máscara para cambiar de color. Los registros de color (a partir de 708 = \$2C4) se combinan con las direcciones 78 y 79 mediante una operación EOR (excluyendo OR).

30 \$50 TMPCHR

Registro temporal utilizado por el motor del display para enviar datos a la pantalla o leerlos de ella.

81 \$51 HOLD1

Igual que TMPCHR. También se utiliza para retener el número de entradas del list del display.

82 \$52 LMARGN

Valor de columna para el margen izquierdo de la emisión en pantalla. 0 indica la columna del extremo izquierdo. El valor default es 2. Vea también RMARGN.

83 \$53 RMARGN

Valor de columna para el margen derecho de la emisión en pantalla. 39 indica la columna del extremo derecho. El valor default es 39. Ambos registros de margen pueden activarse (con ciertas limitaciones) a cualquier valor. De esta manera el editor de pantalla responde si después de 82 y 83 se hace un POKE del mismo valor. Cargue cualquier programa en la memoria, haga POKE 82,n y POKE 83,n (n de 0 a 39) y haga LIST. ¿Qué ocurre?

Más experimentos: el margen derecho está a la izquierda del margen izquierdo, por ejemplo POKE 82,37 y POKE 83,5. O el margen derecho toma un valor superior al margen derecho; POKE 82,37 y POKE 83,47. Sólo si da un valor superior a 39 al margen izquierdo, pulsará seguidamente y con satisfacción la tecla RESET.

RESET renueva los valores default de los márgenes. El borrado o insertado de líneas no será afectado por las modificaciones de los márgenes de salida. Insert Line inserta siempre una línea física, independientemente de su longitud, y Delete Line siempre borra una línea lógica.

Los márgenes también influyen en la identificación acústica del final de la línea lógica. Esta identificación acústica se rige por el final de la línea lógica, que también se reduce si se acortan las líneas físicas. El zumbador siempre suena, pues, antes de finalizar la tercera línea.

84 \$54 ROWCRS

Actual posición de línea del cursor gráfico o de texto. Según el modo gráfico se admiten valores de 0 (arriba) a 19 o 191 (abajo).

85,86 \$55,\$56 COLCRS

Actual posición de columna del cursor gráfico o de texto. Según el modo gráfico se admiten valores de 0 (izquierda) a 39 ó 319 (derecha). El siguiente programa salta con el comando POSITION hacia diversas posiciones de pantalla e imprime allí el valor actual del cursor. El primer carácter de la impresión se sitúa en el lugar del salto y los demás caracteres siguen en el sentido de escritura:


```

0 REM ADR84.BEC
1 REM *****
2 REM * *
3 REM * PROGRAMA DEMOSTRACION DEL *
4 REM * POSICIONAMIENTO DEL CURSOR *
5 REM * *
6 REM *****
10 PRINT CHR$(125):POKE 82,0
20 FOR X=0 TO 30 STEP 6
30 FOR Y=0 TO 22 STEP 2
40 POSITION X,Y:PRINT PEEK(85);",";PEEK(84);
50 NEXT Y
60 NEXT X
70 GOTO 70

```

40: La dirección 86 contiene el byte HI de las posiciones de columna del cursor. Siempre es 0, excepto en GRAPHICS 8, donde puede tomar el valor 1. Por ello basta aquí con leer 85.

Los contenidos de los registros se imprimen en orden inverso, ya que todos los comandos referidos a las posiciones de pantalla indican los valores en el orden columna,línea, o sea: PLOT columna,línea o POSITION X,Y. Puesto que el registro 85 contiene el valor de columna se imprimirá antes de la coma.

70: Evita el fin del programa con READY.

Cabe mencionar que no tiene ningún sentido entrar esta instrucción:

```
POSITION n,m:? PEEK(85);" ";PEEK(84)
```

Esta instrucción no encontrará nunca el valor n,m, puesto que si al final de esta entrada directa se pulsa la tecla RETURN, el cursor salta inmediatamente al principio de la siguiente línea física y éste será precisamente el valor de posición que la instrucción encontrará.

El comando LOCATE en BASIC no sólo inspecciona la posición de pantalla a la que nos hemos dirigido, sino que también mueve el cursor un espacio a la derecha con el siguiente comando PRINT o PUT, modificando los valores en ROWCRS y COLCRS. Por cierto, CHR\$(253), el sonido de alarma (BEL), no es un carácter de presión y por lo tanto no incide en la posición del cursor.

Puesto que el comando LOCATE tiene sus peculiaridades, a continuación le ofrecemos un programa demostrativo:

```
0 REM ADR85.BEC
1 REM *****
2 REM * *
3 REM * INFLUENCIA DE LOCATE *
4 REM * SOBRE EL CURSOR *
5 REM * *
6 REM *****
8 LIST 110
10 GRAPHICS 0:PRINT CHR$(125)
20 POSITION 30,20
30 PRINT PEEK(85);",";PEEK(84);
40 LOCATE 4,4,D
50 PRINT PEEK(85);",";PEEK(84);
60 LOCATE 0,0,X
70 POSITION 10,10:PRINT "A"
80 LOCATE 10,10,D:PRINT D
90 FOR Z=0 TO 1000:NEXT Z
100 LOCATE 10,10,D:PRINT D;" "
110 FOR Z=0 TO 1000:NEXT Z
120 LOCATE 10,10,D:PRINT D;" "
200 GOTO 90
```

10; Aunque no hace falta llamar GRAPHICS 0 expresamente, el comando es necesario en este caso porque LOCATE sólo surge efecto cuando le precede un comando GRAPHICS.

20: Salta hacia la posición 30,20. Por ello los correspondientes valores se encuentran naturalmente en los registros del cursor que en la línea

30: son leídos y emitidos en pantalla. Por favor, tenga en cuenta el punto y coma al final de la instrucción. Este evita que el cursor salte hacia la línea siguiente después de la salida PRINT, manteniéndolo directamente detrás de la impresión. Sin embargo, ya que en línea

40: hay un comando LOCATE, el cursor se mantiene ópticamente detrás de la posición 30,20, mientras que en realidad se encuentra en 4,4. LOCATE localiza el valor COLOR (punto gráfico) o ATASCII (carácter) y lo guarda en la variable indicada.

50: Se vuelve a imprimir el registro del cursor. La coma al final de la instrucción provoca un salto tabulador.

60: Con el salto producido en esta línea vuelve a quedar atrás un cursor óptico.

70: Con PRINT se imprime una A en el lugar 10,10 para otra experiencia.

80: LOCATE encuentra en 10,10 el valor ATASCII de A (=65), que se imprime inmediatamente. ¡El comando PRINT directamente detrás de LOCATE modifica el valor en el lugar localizado!

90: Un pequeño descanso para la vista.

100: Esta vez, el comando LOCATE ya no localiza el valor 65 (A) sino 32 (espacio) en 10,10.

110: Otro momento de descanso.

120:Otra vez se ha modificado el valor de este lugar mediante LOCATE y a continuación PRINT. Ahora se ve un espacio inverso y se localiza el valor 160.

200:Este juego sigue indefinidamente hasta que se Interrumpa con BREAK.

El truco con LOCATE sólo funciona en GRAPHICS 0. En los ATARI BASIC reference Cards se asegura, que también funciona de la misma manera en GRAPHICS 9,10 y 11. Sin embargo, el programa de ejemplo adjunto no produce el efecto anunciado, puesto que al conectar un modo gráfico sin ventana de texto, el comando PRINT cambia inmediatamente a GR.0.

87

\$57

DINDEX

Los cuatro bits inferiores recogen el número de identificación del modo gráfico activado. Puesto que en muchas funciones gráficas el OS se refiere a este registro, éste puede utilizarse para simular un modo gráfico distinto al que realmente ha sido llamado con GRAPHICS. El registro puede tomar valores de 0 a 15 que corresponden a los comandos GRAPHICS y no a los comandos ANTIC (vea apéndice display list):

El programa siguiente le muestra la forma de simular nuevos modos gráficos con ayuda de DINDEX, que crea una resolución gráfica de 160 (de 0 a 159) por 192 (de 0 a 191) puntos de pantalla. Cada pixel ocupa, pues, dos puntos de pantalla y cada línea del modo equivale a una línea del TV. Los colores se definen tal como está previsto para GRAPHICS 8 (o sea 24). Aunque pueden llamarse tres valores COLOR al igual que para GRAPHICS 7 (o sea 23), sólo aparecerán diferentes en cuanto a su claridad:

```

0 REM ADR87.BEC
1 REM *****
2 REM * *
3 REM * GRAFICOS 7 1/2 *
4 REM * *
5 REM *****
10 GRAPHICS 24
20 POKE 87,7
30 COLOR 3
40 FOR X=0 TO 159:PLOT X,0:NEXT X
50 COLOR 2
60 FOR Y=0 TO 95:PLOT 0,Y:PLOT 159,Y:NEXT Y
70 COLOR 3
80 PLOT 0,0:DRAWTO 159,95
90 COLOR 2
100 PLOT 0,95:DRAWTO 159,0
110 COLOR 1
120 FOR X=0 TO 159:PLOT X,95:NEXT X
200 POKE 89,PEEK(89)+15
210 COLOR 3
220 FOR X=0 TO 159:PLOT X,0:NEXT X
230 COLOR 2
240 FOR Y=0 TO 95:PLOT 0,Y:PLOT 159,Y:NEXT Y
250 COLOR 1
260 PLOT 0,0:DRAWTO 79,95:DRAWTO 159,0
270 PLOT 0,95:DRAWTO 79,0:DRAWTO 159,95
280 COLOR 2
290 FOR X=0 TO 159:PLOT X,95:NEXT X
300 GOTO 300

```

10: Se llama al modo gráfico 8 sin ventana de texto.

20: En DINDEX se deposita el valor 7.

30 a 120: Dibujan diversas líneas de "diversos colores". Naturalmente, en lugar de los PLOT FOR-NEXT también pueden escribirse DRAWTO PLOT.

Tenga en cuenta que el gráfico 8 llamado admite, en principio, 320 PPL (pixel per line = puntos de pantalla por línea), pero a causa del POKE 87,7 la línea ya se ha llenado hasta la mitad (159). Estos son los PPL de GRAPHICS 7. Puesto que ahora se dispone de dos bits de información de color para cada pixel (vea apéndice memoria de pantalla para más información), podemos dirigirnos a tres valores COLOR.

Con el comando GRAPHICS se llama al display list. En GRAPHICS 8 cada línea del modo es tan alta como una línea del TV. La pantalla será tratada, pues, en 191 líneas de modo. Sin embargo, si se llama una posición de pantalla a través de PLOT o DRAWTO, que sobrepasa los valores admitidos de GRAPHICS 7, entonces se produce ERROR-141 (posición del cursor incorrecta), ya que en este lugar empieza a actuar el valor modificado con POKE en DINDEX. Esto significa que con los valores de líneas y columnas de GRAPHICS 7 sólo puede tratarse la mitad superior de la pantalla. ¿Qué hacer?

Mediante el comando GRAPHICS 8 se ha reservado memoria suficiente como para recoger los datos de imagen de toda la pantalla. Sin embargo, las manipulaciones realizadas impiden ahora escribir comandos BASIC con efectos gráficos en la mitad inferior de las celdas de memoria.

Todos los comandos que hacen referencia a las posiciones de pantalla calculan internamente el offset del byte SM, que contiene los datos del punto buscado. Esto se hace según la fórmula dirección inicial SM + valor de columna/PPB (pixel per byte) + valor de línea*BPL (bytes per line). Por ello, las coordenadas de GRAPHICS 7 sólo admiten un offset máximo de $40 + 95*40 = 3840$.

200: Para poder acceder a los datos de imagen de la mitad inferior de pantalla, el vector hacia la dirección inicial SM se incrementa en los 3840 bytes hallados anteriormente.

210 a 290: Con ayuda de los valores de posiciones admitidos, los puntos gráficos pueden colocarse ahora con PLOT en la mitad inferior de la pantalla.

300: Evita el fin del programa. Puesto que no se dispone de ninguna ventana de texto, se conectaría GRAPHICS 0, borrando por lo tanto el gráfico construido, simplemente para que el ordenador pueda mostrar su READY.

Si ahora se ha despertado su curiosidad para conocer todos los trucos realizables en el campo de los gráficos del ATARI, satisfágala modificando ADR87.BEC en las líneas siguientes:

```
0 REM ADR87A.BEC
1 REM *****
2 REM *
3 REM * UN GRAFICO APETITOSO *
4 REM *
5 REM *****
10 GRAPHICS 24:A=2
15 TRAP 15:A=A+1
20 POKE 87,A
300 GOTO 15
```

(vea también los apéndices display list y memoria de pantalla).

88,89 \$58,\$59 SAVMSC

Puntero hacia la dirección inicial de la memoria de pantalla (SM = screen memory). En este byte se encuentran los datos de los pixel a visualizar en la esquina superior izquierda de la pantalla.

Los datos gráficos ocupan la memoria desde SAVMSC hasta RAMTOP (40959 = \$C0000). Con la llamada de un comando GRAPHICS el puntero se ajusta hacia el correspondiente lugar de la memoria. Sin embargo, el usuario puede modificar el puntero según sus necesidades.

Es posible, por ejemplo, depositar en la memoria los datos de varios contenidos de pantalla y modificar más tarde el contenido de la pantalla, repentinamente, ajustando el puntero SM hacia las diversas direcciones iniciales. Esto sólo puede hacerse después de cada VBLANK, cuando se vuelve a leer el display list desde el principio, pero esto ocurre cincuenta veces por segundo.

En el siguiente programa se ha elegido GRAPHICS 3 (19) porque éste es un modo que tiene la agradable característica de que la SM sólo consta de 240 bytes, o sea menos que una página. De esta manera se pueden depositar, tranquilamente, los datos de cada pantalla en páginas consecutivas de la memoria y mover rápidamente el gráfico por la pantalla al modificar el byte HI del puntero de SM. Sin embargo, modificar SAVMSC no causa ningún efecto en este caso, puesto que el puntero de SM también forma parte del display list, y el procesador de video extrae de ese puntero en el DL el lugar de la memoria donde debe ir a buscar los datos de pantalla:

```
0 REM PAGING,BEC
1 REM *****
2 REM *
3 REM * CONMUTACION DE PANTALLA *
4 REM *
5 REM *****
10 D=1
20 GRAPHICS 19
30 POKE 708,50
40 POKE 709,52
50 POKE 710,54
60 FOR P=0 TO 20
70 COLOR C+1
80 X=INT(RND(0)*12)
90 Y=INT(RND(0)*12)
100 PLOT 19-X,11-Y
110 PLOT 19+X,11-Y
120 PLOT 19+X,11+Y
130 PLOT 19-X,11+Y
140 PLOT 19-Y,11-X
150 PLOT 19+Y,11-X
160 PLOT 19+Y,11+X
170 PLOT 19-Y,11+X
180 NEXT P
```



```

200 D=D+1:C=C+1:IF C=3 THEN C=0
210 IF D=7 THEN 300
220 SM=PEEK(88)+PEEK(89)*256
230 FOR K=0 TO 239
240 POKE SM-D*256+K,PEEK(SM+K)
250 NEXT K
260 GOTO 20
300 DL=PEEK(560)+PEEK(561)*256
310 Z=PEEK(DL+5)
320 FOR J=0 TO 6
330 POKE DL+5,Z-J
340 FOR W=0 TO 50:NEXT W
350 NEXT J
360 FOR J=5 TO 2 STEP -1
370 POKE DL+5,Z-J
380 FOR W=0 TO 50:NEXT W
390 NEXT J
400 GOTO 320

```

30 a 50: Los valores de color para el comando COLUR se depositan en los registros de color (vea COLOR0, 708 = \$2C4 y siguientes):

60 a 180: La presentación. El programa crea unos cuantos gráficos aleatorios que van cambiando de color.

220 a 250: Los bytes de datos son leídos uno tras otro en la memoria de pantalla (PEEK SM+K) y escritos en el mismo orden (POKE SM - D*256 + K) en una página superior (SM - D*256). D determina el número en que aumentan las páginas donde se escribirán con POKE los datos correspondientes, y K cuenta todos los 240 bytes de GRAPHICS 3.

300: Calcula la dirección inicial del display list (vea SDLSTL, 560,561 = \$230,\$231).

310: El byte HI del puntero de SM en el DL es el quinto byte (¡contando el byte cero!).

320 a 390: Trasladan el vector de SM cada vez que se ejecuta un corto bucle de espera. De esta manera, los cinco gráficos creados en la parte superior del programa cambian repentinamente en la pantalla.

Usted podrá imaginarse sin dificultad cómo pueden crearse animaciones, al igual que en las películas de dibujos animados. Mientras sólo se utilice el modo gráfico 3 y el propio programa en BASIC no sea demasiado largo, se dispone de espacio suficiente (en el 800) para unos cien contenidos de pantalla. Si nos limitamos a una sucesión de imágenes con destellos de 16 proyecciones por segundo, este ordenador le permitirá pasar ocho segundos de cine en tiempo real.

(encontrará más juegos gráficos en el apéndice memoria de pantalla).

90 \$5A OLDROW

Línea anterior del cursor gráfico. Es renovado por ROWCRS (84 = \$54) antes de que éste tome un nuevo valor. Así, DRAWTO o el comando FILL (XIO 18) disponen de las coordenadas del comienzo en OLDROW y de las coordenadas de destino en ROWCRS.

91,92 \$5B,5C OLDCOL

Función parecida a OLDROW en referencia a las posiciones de columnas, por lo que ocupa dos bytes (320 columnas en GRAPHICS 8):

93 \$5D OLDCHR

Contiene el valor del carácter que se encuentra bajo el cursor y se utiliza para actualizarlo al mover este último.

94,95 \$5E,\$5F OLDADR

Contiene la dirección de la memoria de pantalla que corresponde a la posición actual del cursor. Se utiliza junto con OLDCHR para actualizar el carácter cuando el cursor está en movimiento.

96 \$60 NEWROW

Coordenadas de las líneas del punto hacia el cual debe dirigirse DRAWTO o XIO 18 (FILL).

97,98 \$61,\$62 NEWCOL

Coordenadas de las columnas del punto hacia el cual debe dirigirse DRAWTO o XIO 18. NEWROW y NEWCOL reciben su valor de ROWCRS y COLCRS (84 - 86 = \$54 - \$56), a fin de que estos registros puedan tomar las coordenadas siguientes del cursor durante la rutina DRAWTO o FILL.

99 \$63 LOGCOL

Indica la posición del cursor dentro de la línea lógica. En el ATARI, una línea lógica puede tener una longitud de hasta tres líneas de GRAPHICS 0, o sea $3*40 = 120$. LOGCOL puede tomar por lo tanto valores de 0 a 119. Este registro es utilizado por el motor del display por ejemplo para emitir

la señal acústica de "línea llena".

100-105 \$64-\$69 -

Depósito temporal utilizado por el motor del display para almacenar valores diversos.

106 \$6A RAMTOP

Indica el RAM de aplicaciones disponible en pages (=256 bytes).

PEEK(106)*256 determina la máxima dirección libre.

Esta dirección puede utilizarse para crear una zona protegida de la memoria por ejemplo para rutinas en lenguaje máquina, juegos de caracteres libremente definidos, datos de player-missile o memorias de pantalla alternativas. La siguiente instrucción reserva la zona deseada:

POKE(106),PEEK(106)-n

n es el número de páginas de la memoria a reservar. El valor PEEK(106)*256 no puede ser nunca inferior que PEEK(144)+PEEK(145)*256 (MEMTOP).

Cuando se cambia RAMTOP para reservar memoria, debería seguir inmediatamente un comando GRAPHICS. Para ello se puede llamar tranquilamente al modo gráfico que en este momento está activado. Ello provoca que los display list y los data gráficos se muevan siguiendo al nuevo RAMTOP.

(≠) Un comando GRAPHICS o CLEAR borra 64 bytes por encima de RAMTOP. Mediante el scroll de la ventana de texto en el modo gráfico se escribe encima de 800 bytes por encima de RAMTOP, ya que en realidad la ventana de texto efectúa el scroll de la totalidad de los datos de pantalla de GR.0 que son,

además de las cuatro líneas de dicha ventana, otras 20 líneas de 40 bytes cada una (= 800 bytes). Cuando hay que reservar memoria deben considerarse estos efectos para blindar en consecuencia una zona mayor, de manera que detrás de los datos a proteger aún quede una zona de seguridad lo suficientemente grande.

Cuando se trabaja con GRAPHICS 7, 8, 9, 10, 11, 14 o 15, pueden producirse perturbaciones al desplazar RAMTOP de forma que los display list y los datos de SM crucen un límite de 4k. Si aparecen cualquier tipo de resultados oscuros en la salida a pantalla puede dar resultado reducir RAMTOP en un múltiplo de 16 (4*4 páginas = 4k).

Otra posibilidad de proporcionar una zona protegida de memoria se encuentra por debajo de MEMLO (743,744 = \$2E7,\$2E8).

107 \$6B BUFCNT

Es utilizado por el editor de pantalla como contador para el resto de la línea lógica.

108,109 \$6C,\$6D BUFSTR

Depósito temporal. Devuelve el carácter al que se apunta con BUFCNT.

110 \$6E BITMSK

Utilizado por el motor del display OS como máscara bit en las rutinas bit-mapping. Además se utiliza como memoria intermedia.

111 \$6F SHFAMT

Ajuste de pixel. Calcula la posición de un pixel dentro de un byte de información gráfica.

112,113 \$70,\$71 ROWAC

Acumulador para controlar las líneas al realizar el PLOT.

114,115 \$72,\$73 COLAC

Acumulador para controlar las columnas.

116,117 \$74,\$75 ENDPT

Punto final de la línea a trazar. Controla el proceso del PLOT de puntos en una línea.

118 \$76 DELTAR

Diferencia de líneas = el valor absoluto de NEWROW - ROWCRS.

119,120 \$77,\$78 DELTAC

Diferencia de columnas = valor absoluto de NEWCOL - COLCRS.
Se utilizan DELTAR y DELTAC para calcular el incremento de la línea con la que debe realizarse el PLOT.

121,122 \$79,\$7A KEYDEF

Puntero hacia la tabla de definiciones para convertir el código de teclado en valores ATASCII. (≠: vea 760 y 761 = \$2F8 y \$2F9)

123 \$7B SWPFLG

Flag para controlar el cursor en la pantalla partida (con ventana de texto).

124 \$7C HOLDCH

Aquí se deposita el valor de un carácter antes de ejecutar la lógica CONTROL o SHIFT para él.

125 \$7D INSDAT

Memoria intermedia utilizada por el motor del display para el carácter que se encuentra bajo el cursor y para reconocer el final de línea.

126,127 \$7E,\$7F COUNTR

Comienza con el mayor valor o de DELTAR o de DELTAC. Corresponde al número de puntos con los que debe realizarse el PLOT para dibujar una línea. Disminuye en 1 con cada punto con el que se ha efectuado el PLOT. Si este byte vale 0, la línea está terminada.

Puntero hacia la mínima dirección disponible para el programa en BASIC. Los primeros 256 bytes sirven de buffer para la salida de token. Un token es un equivalente numérico de comandos, operadores y funciones, del tamaño de un byte.

El valor se obtiene aquí de MEMLO durante la inicialización del ordenador o durante un comando NEW, pero no durante el RESET. Esto significa que al cambiar MEMLO también debe adaptarse el valor de LOMEM.

Al grabar un programa con SAVE, se graban en primer lugar los siete punteros BASIC desde LOMEM hasta STARP. Por lo tanto se resta el valor de LOMEM de todos estos punteros de 2 bytes. De esta forma, los dos primeros bytes grabados son ceros. Después de los punteros se graban la tabla de los nombres de variables y sus valores y a continuación el programa BASIC convertido en forma de tokens.

Al efectuar LOAD, el valor de MEMLO se suma a cada uno de los siete punteros y a continuación se graban estos valores en las direcciones de los punteros en la página cero. Después se reservan 256 bytes por encima de MEMLO para el buffer de salida de los token, y seguidamente a este buffer se almacena el programa BASIC.

Naturalmente con estos siete punteros se pueden causar efectos desastrosos. Cuando tropas enemigas invaden el país, todos los indicadores de caminos y calles se quitan o se les da la vuelta. Cuando se teme la invasión de fisgones en el propio y valioso programa, se les puede "dar la vuelta" a esos punteros con el fin de despistar en algo al invasor. Pero no cante victoria todavía, porque ningún "fusilador" de verdad se deja intimidar por estas "criaturadas". Esto quiere decir que las manipulaciones en ese sentido sólo ofrecen una protección limitada.

Tal como ya mencionamos anteriormente, con SAVE se graban también los valores de los punteros. Sin embargo, no ocurre lo mismo haciendo LIST. Por esta razón, los cambios realizados en los punteros en BASIC ofrecen en primer lugar una protección de listado. Aunque todavía pueda copiarse el programa sin dificultad con SAVE y LOAD, los listados muestran, sin embargo, un aspecto más o menos exótico.

El siguiente programa muestra cómo se puede crear algo de confusión con una sola línea en BASIC:

```

0 REM ADR128.BEC
1 REM *****
2 REM * *
3 REM * LISTADO ENLOQUECIDO *
4 REM * *
5 REM *****
10 A=A+10
20 B=10:C=5:D=7.5
30 F=A+B:G=A*C:H=A/D
40 PRINT F,G,H
50 GOTO 10
100 REM X1033,#1,0,0;"D:SECRETO"
110 REM Y=PEEK(128)+PEEK(129)*256+3:POKE
128,Y-INT(Y/256)*256:POKE129,INT(Y/256):SAVE "D:SECRETE"
120 REM RUN"D:SECRETO"

```

```

0 REM ADR128.BEC
1 REM *****
2 REM *                               *
3 REM *   LISTADO ENLOQUECIDO       *
4 REM *                               *
5 REM *****
10 ;PF=:PF+10
20G=10:H=5:%%***** =7,5
30 D=:PF+G:Y=:PF*H:ADR128.BEC
40 PRINT D,Y,ADR128.BEC
50 GOTO 10
100 REM X1033,#1,0,0"D:SECRETE"
110 REM Y=PEEK(128)+PEEK(129)*256+3:POKE
128,Y-INT(Y/256)*256:POKE129,INT(Y/256):SAVE"D:SECRETE"
120 REM RUN"D:SECRETE"

```

10 a 50: Estas líneas contienen un programa BASIC totalmente incoherente, programa que debería ser protegido sobre todo contra miradas indiscretas ya que utiliza una serie de técnicas de programación que alteran la programación BASIC.

110: Borre tanto el número de línea como el REM y escriba esta línea en modo directo. Se creará un fichero en el diskette cuyo nombre será "SECRETE". Durante el proceso se irán restando de todos los punteros el disparatado valor de LOMEM, tal y como se explicó anteriormente.

120: Borre ahora el número de línea y el REM para realizar una entrada en modo directo. Por supuesto también puede ordenar primero LOAD y después RUN para conseguir mayor efectividad. El resultado que se muestra en la pantalla le permite comprobar claramente que el programa trabaja sin errores. ¿Y por qué no? Pero quizás ahora desee efectuar un

LIST en el ordenador. ¿Qué le parece este LIST?

Por cierto, también puede probar con LIST"C: o LIST"D:; ¡pero el ordenador disfruta especialmente con un LIST"P:"!

100: Para borrar este confuso programa de su diskette tan sólo debe entrar directamente esta línea (eliminar número de línea y REM y pulsar RETURN). Esta instrucción le evita tener que acceder al DOS y efectuar la opción D y, si no hay ningún MEM.SVE en el diskette, de cargar además nuevamente el programa de "D:" en el ordenador después de la opción B.

130,131 \$82,\$83 VNTP

Este puntero apunta hacia el primer byte de la tabla que contiene los nombres de variables. Los nombres de las variables se almacenan según el orden en que éstas son utilizadas por el usuario. Cada vez que usted cometa un error de teclado y la configuración del mismo coincida casualmente con una de las variables, este error también constará en la tabla de variables.

Los nombres de las variables se almacenan según formato ATASCII. Las variables indexadas son anotadas hasta llegar al paréntesis abierto, por el cual serán identificadas. Las variables de cadena (strings) terminan con el carácter \$. En teoría, los nombres de variables pueden tener la misma longitud que una línea lógica. El ATARI también es capaz de distinguir entre todas estas variables. Esto quiere decir que, a diferencia de otros ordenadores personales, el ATARI no considera sólo los dos o tres primeros caracteres de cada variable. Para reconocer dónde termina un nombre de variable y dónde comienza el siguiente se activa el BMS (bit más

significativo = bit 7) de valor decimal 128 con el último carácter de la variable. Para el editor de pantalla, esto significa emitir el carácter inverso (negativo).

En esta tabla pueden registrarse hasta 128 nombres de variables. Esto puede resultar insuficiente si se pretende escribir un programa muy largo. A continuación le enseñamos algunos trucos cuyo fin es evitar este problema:

1: Elija nombres de variables lo más breves posible. Los nombres largos únicamente ocupan mucha memoria y hacen que el programa sea más lento. Sin embargo, los nombres de variables largos pueden memorizarse con mayor facilidad durante la confección de un programa. Por ello es aconsejable escribir el programa tranquilamente utilizando términos claros para designar variables, sustituyéndolos una vez finalizado el programa. ¡No! Naturalmente no nos referimos a tener que sustituirlos en el mismo programa -línea por línea y pasando por alto uno por aquí y otro por allá, enfrentándonos durante horas a los ERRORes- sino a sustituirlos directamente en la tabla de variables.

2: Si no queda espacio libre en la tabla de variables, puede ser útil dar un pequeño rodeo. Es muy probable que con la afición de programar se nos haya deslizado algún que otro nombre en la tabla, el cual ya no sea necesario. Grabe el programa en el diskette con LIST, entre NEW -que también borra la tabla- y vuelva a entrarlo con ENTER. Ahora se han borrado todos los nombres de las variables inútiles. Pero para ello debe efectuar un LISTado, puesto que SAVE graba la tabla entera y la vuelve a cargar.

3: Utilice variables auxiliares varias veces. No hay ninguna razón para mencionar en el primer bucle FOR-NEXT, una determinada variable y otra distinta en el siguiente. Puede utilizar tranquilamente el mismo contador para todos los bucles FOR-NEXT que no estén indentados entre sí.

4: Si su programa no padece de escasez de variables, es aconsejable asignar una variable a todos aquellos valores numéricos que son utilizados más de tres veces a lo largo del programa. Por un lado, una variable ocupa memoria en la tabla de nombres y por otro en la de valores de variables. En el propio programa BASIC la variable ocupa tan sólo el mismo número de bytes como caracteres tiene su longitud, o sea un único byte en el caso más favorable, mientras que ¡cada valor numérico ocupa siempre seis bytes! Escriba pues GOTO J en vez de GOTO 300, por ejemplo. Esto nos indica claramente que el espacio de memoria ahorrado al utilizar variables, es directamente proporcional al número de veces que cierto valor numérico está contenido en un programa. Solamente aparecen problemas si usted pretende reenumerar posteriormente las líneas de un programa de este tipo utilizando una utilidad renumber, puesto que no hay ningún programa "renum" que convierta destinos de salto alfanuméricos.

Ahora observe la tabla de variables:

```
0 REM ADR130.BEC
1 REM *****
2 REM * *
3 REM * LISTAR TABLA DE VARIABLES *
4 REM * *
5 REM *****
10 A=PEEK(130)+PEEK(131)*256:E=PEEK(132)+PEEK(133)*256:FOR
I=4 TO E-A-1:O=PEEK(A+I):IF O>128 THEN O=O-128:GOTO 30
20 PRINT CHR$(O);:GOTO 40
30 PRINT CHR$(O),:GOTO 40
40 NEXT I:END
100 B=1:C=2:D=3:DIM F$(1),G(2),H$(3)
200 REM V=PEEK(130)+PEEK(131)*256+4:POKE V+X,CHR+128
300 REM X=0:CHR=84:REM B SERA T
```

10: La variable A recoge el puntero hacia el principio de la tabla de nombres de variables (direcciones 130,131). Con E se designa el puntero hacia el final de la misma (direcciones 132,133). El siguiente bucle FOR-NEXT lee ahora todos los bytes que se encuentran entre A y E. Puesto que este pequeño programa destinado a listar los nombres de las variables también contiene sus propias variables (A, E, I y O), el bucle comienza a partir del valor 4. Pruebe sustituyendo aquí un 1 o un 0. La condición IF comprueba si el byte leído corresponde al último carácter del nombre de una variable. Únicamente en caso afirmativo el programa salta hacia la línea 30 donde la coma proporciona un salto tabulado, separando así las variables. .

20: El comando CHR\$ basta para hacer que las variables sean legibles, ya que los nombres de variables se almacenan en la tabla según formato ATASCII. El punto y coma enlaza los caracteres directamente uno tras otro.

30: Igual que la línea 20, pero con salto TABulado.

100: Esta línea representa el programa BASIC que en este caso sólo consta de seis variables.

300: Si ahora se identifica la posición exacta de una variable en la tabla, efectuando un POKE del valor ATASCII correspondiente, se puede cambiar el contenido de la celda de memoria y con ello el nombre de la variable. La primera variable (o sea la variable cero) del sustituto de un programa BASIC en la línea 100 es B. Esta variable debe tomar ahora el nombre "T". El valor ATASCII de T es 84. Quite el número de línea y el REM y escriba la instrucción en modo directo. X designa el lugar que debe ser cambiado. Debe contarse cada carácter y no sólo cada variable. B es el carácter número 0 de la tabla, y X toma el valor 0. A CHR se asigna el valor ATASCII del nuevo carácter.

200: En esta línea se efectúa el cambio de variables. Con V se designa el puntero (VNTP) hacia el principio de la tabla de variables. En este lugar aún se suma el valor 4 porque el programa precedente contiene cuatro variables en las líneas 10 a 40, que serán salvadas de esta forma. Por cierto, el primer byte al que apunta el VNTP siempre es 0. Si hace que éste se imprima a través de la rutina indicada, aparecerá un corazón delante (gracioso, ¿no?).

En la dirección V (primer carácter de la primera variable de la tabla) + X offset del carácter a modificar se coloca mediante POKE el valor ATASCII del nuevo carácter. En nuestro ejemplo, éste corresponde al último de la variable, por lo que debe activarse además el bit 7 (CHR+128) para identificar el final de la misma.

Borre también de esta línea su número y el REM para realizar una entrada en modo directo. A continuación escriba RUN para volver a imprimir la tabla de los nombres de variables, y verá ...

Si ahora quiere convertir F\$ en V\$, indique un 3 en el lugar de X y un 86 en el de CHR, pulse RETURN y zasss.

UTILIZANDO estas líneas como base, se puede confeccionar fácilmente un programa de utilidades que nos permita cambiar los nombres de las variables a través de confortables entradas para X y CHR y consultar si se trata de un carácter final o no. Es aconsejable escribir un programa de este tipo con números de línea muy elevados (de 32700 a 32767), de manera que en caso necesario pueda ser cargado detrás de un programa BASIC ya existente en el ordenador. Para ello también hay que tener en cuenta que, entre el programa fuente y el de utilidad, no se produzcan cruces con los nombres de las variables.

La acción de alargar los nombres de las variables sólo podría resolverse con grandes dificultades, puesto que comportaría desplazar hacia abajo todas las que se encuentran en las posiciones siguientes de la tabla. Sin embargo, es fácil recortar los nombres de las variables. Sólo hace falta POKEar 0 en las posiciones correspondientes, o en su caso 128 si el carácter a eliminar es el último. Si bien es cierto que únicamente con esta acción no se han reducido todavía los registros en la tabla de variables, ya que aún se ocupa el mismo número de bytes, también puede resolverse con LIST"D:", NEW, ENTER"D:".

Si en algún momento le apetece ocultar de nuevo sus obras de programación de indiscretas miradas LIST, también dispone de la tabla de variables. Qué le parece, por ejemplo, un listado en el que todas las variables sean idénticas :


```

0 REM ADR131.BEC
1 REM *:*****:*****:*****
2 REM *
3 REM *DESTRUIR TABLA DE VARIABLES *
4 REM *
5 REM *****:*****
10 A=PEEK(130)--P_EK(131)*256
20 Z=PEEK(132)+PEEK(133)*256
30 FOR J=A TO Z:POKE J,129:NEXT J
100 B=10:C=30:D=2.5
110 F=C-B:G=B/D:H=D*C
120 ? F,G,H

```

10: El puntero VNTP que apunta hacia el principio de la tabla que contiene los nombres de las variables es asignado a A.

20: El puntero VNTD hacia el final de la tabla de los nombres de las variables es anotado en Z.

30: Ahora ya sólo le queda llenar todos los bytes que se encuentran entre A y Z con un mismo valor ATASCII.

Escoja un valor ATASCII más 128 para sustituir por el mismo carácter todas las variables del listado. También es muy bonito efectuar un POKE con valor 128, provocando de esta forma la desaparición de todas las variables del listado. También valores inferiores a 128 resultan asimismo muy graciosos, puesto que con ellos ya no se pueden identificar los finales de dichas variables. Observe usted el aspecto tan divertido que adquiere un listado de este tipo. Pero la culminación del sustituto de las variables es el bonito número 155. Este es el valor ATASCII correspondiente de RETURN. De esta forma, cada variable se sustituye por un RETURN. Incluso en programas más bien cortos, el resultado de tal operación son unos listados maravillosamente largos que no contienen ni una sola variable.

¡Que se divierta con este juego de confusiones!

132,133

\$84,\$85

VNTD

Este puntero apunta hacia el final de la tabla que contiene los nombres de las variables. Si existen 128 nombres en la tabla, el puntero apunta hacia el primer byte detrás del último carácter de la última variable. Si hay menos variables en la tabla, el puntero apunta hacia un byte vacío (0; ATASCII 0 = corazón) que se encuentra inmediatamente detrás de la última variable, o sea en el lugar donde se colocará el próximo nombre.

Otra rutina eficaz de protección contra el listado consiste en cambiar los dos punteros VNTP y VNTD. El valor a colocar con POKE en los bytes de los punteros sólo depende de su fantasía. En el siguiente ejemplo se ha escogido un 0:

```

0 REM ADR132.BEC
1 REM *****
2 REM *
3 REM *   DESTRUIR APUNTADOR DE
4 REM *   LA TABLA DE VARIABLES
5 REM *
6 REM *****
10 FOR J=0 TO 3:POKE 130+J,0:NEXT J
100 A=10:B=30:C=7.5
110 F=A+B:G=B/C:H=C*A
120 PRINT F,G,H

```

10: Esta línea que parece tan inocente basta para confundir los dos punteros.

100 a 200: Este sustituto de un programa BASIC tan sólo está aquí para que usted pueda apreciar las impresionantes consecuencias del POKE confuso al efectuar un LISTado.

134,135 \$86,\$87 VVTP

Puntero hacia la tabla de valores de las variables. Aquí se anotan los valores actuales de todas las registradas en la tabla de nombres. Cada variable ocupa ocho bytes en la tabla de valores. El primer byte indica el tipo de variable de que se trata.

El número de identificación 00 corresponde a un escalar (valor numérico no dimensionado).

El número de identificación 65 corresponde a un array DIMensionado (variable indexada).

El número de identificación 129 corresponde a un string (variable de cadena).

El segundo byte contiene el número de la variable, que es un número entre 0 y 127.

En el escalar, los seis bytes restantes toman la constante BCD (binary coded decimal = número decimal codificado en binario).

En el array, los byte 3 y 4 indican el offset de STARP (140,141 = \$8C,\$8D), que corresponde a la posición dentro de la tabla del string y del array. Los bytes 5 y 6 contienen el primer DIMensionado + 1, y los bytes 7 y 8 contienen el segundo + 1. Si una variable tiene DIMensión simple, los bytes 7 y 8 tienen valor 0.

En las variables de cadena (string) los bytes 3 y 4 indican igualmente el offset de STARP. Los bytes 5 y 6 contienen la actual longitud del string, mientras que los bytes 7 y 8 contienen la longitud definida con DIM.

Si desea llenar un string con caracteres idénticos, existe para ello una técnica sencilla. Simplemente debe depositar el carácter deseado en la primera posición del string, el propio string en la última posición del mismo, e inmediatamente se llena por completo el string con ese carácter depositado en primer lugar:

```
0 REM ADR135.BEC
1 REM *****
2 REM *                               *
3 REM *      COMPLETAR STRING        *
4 REM *                               *
5 REM *****
10 DIM TX$(266)
20 TX$(1)="?"
30 TX$(266)=TX$
40 TX$(2)=TX$
100 ? TX$
```

10: Se efectúa un DIM.

20: Se deposita el primer carácter.

30: Se deposita el último carácter. ¡Pero hay que depositar TX\$ y no "?!".

40: Ahora se llena la segunda posición y el string estará listo.

100: Aquí puede observar el resultado.

No es obligatorio llenar el string completo de esta forma. También se puede escribir sólo parcialmente en él:

```
0 REM ADR135X.BEC
1 REM *****
2 REM * *
3 REM * EXPLOSION DE MEMORIA *
4 REM * *
5 REM *****
10 DIM TX$(FRE(0)-1)
20 TX$(1)="!"
30 TX$(9999)=TX$
40 TX$(2)=TX$
100 ? TX$
```

10: Se reserva toda la memoria aún disponible (FRE(0)) para TX\$. ¿Y por qué no?

30: Sin embargo, se llenan sólo los primeros 9999 miembros con "!".

De todas maneras, en este programa se presenta un pequeño problema con la forma aquí mostrada. Puesto que TX\$ ocupa toda la memoria, ya no se puede realizar una salida en pantalla. Esto significa que tendremos que limitarnos en la línea 10. Pero es posible que este bonito efecto lleve a los lectores más creativos a una idea de cómo configurar un programa -mediante el llenado de la memoria libre- de tal manera que realmente funcione de la forma prevista, pero registrando regularmente los errores que sobrepasen el límite, realizados por posibles experimentos indeseados del usuario.

Pero volvamos a las cuestiones más serias de la programación. El efecto descrito anteriormente funciona no sólo con un único carácter, sino también con secuencias enteras de caracteres. Para ello se deposita en la posición 1 del string la secuencia de caracteres; a continuación, se deposita el propio string al final del mismo de tal manera, que el último carácter de la secuencia ocupe la última posición a llenar. O sea, si se pretende llenar el string completo, debe restarse la longitud de la secuencia de caracteres de la longitud del string. Finalmente debe colocarse además el string en la posición que sigue a la secuencia de caracteres depositada en primer lugar, o sea en la posición 1 + longitud de secuencia de caracteres. En fin, no quiero extenderme más:

```

0 REM ADR135A.BEC
1 REM *****
2 REM *                               *
3 REM * COMPLETAR PERIODICAMENTE    *
4 REM * UN STIRNG.                  *
5 REM *                               *
6 REM *****
10 DIM TX$(264)
20 TX$(1)="!#?$"
30 TX$(261)=TX$
40 TX$(5)=TX$
100 PRINT TX$

```

10:TX\$ toma la DIMensión 264.

20:En las posiciones 1 a 4 se deposita la secuencia de caracteres.

30:A continuación se coloca TX\$ en las posiciones 261 a 264

40:y después además en posición 5, o sea de 5 a 8.

136,137 \$88,\$89 STMTAB

Puntero hacia la tabla de instrucciones, o sea el propio programa BASIC en forma token. Los dos primeros bytes de una línea BASIC convertida en forma token contienen el número de línea (LO,HI*256). Les sigue un byte contador que indica el número de celdas de memoria ocupadas por la línea BASIC. Este byte contador toma su valor en el momento en que ha finalizado la conversión en forma token, efectuada en el buffer de salida del token (256 detrás de LOMEM). El byte 3 contiene el offset de línea (BASIC) a línea.

Si quiere descubrir en qué celdas de memoria reside su programa BASIC, pruebe el siguiente programa:

```
0 REM ADR136.BEC
1 REM *****
2 REM * *
3 REM * D1.RECCIONES INICIALES/ *
4 REM * NUMEROS DE LINEA BASIC. *
5 REM * *
6 REM *****
10 A=10
20 B=30
30 C=7.5
40 F=A*B
50 G=B/C
60 H=C-A
70 PRINT F,G,H
100 T=PEEK(136)+PEEK(137)*256
110 ZN=PEEK(T)+PEEK(T+1)*256
120 IF ZN>32767 THEN END
130 PRINT "LINEA DE BASIC #";ZN;" TIENE DIRECCION":? "DE
INICIO ";T:?"
140 T=T+PEEK(T+2)
150 GOTO 110
```

10 a 70: Contiene el sustituto de un programa.

100: Calcula el valor numérico del puntero STMTAB.

110: Determina el número de línea ZN a partir de los bytes 0 y 1 de la primera línea BASIC convertida en forma token.

130: El número de línea ZN y la dirección inicial T se muestran en pantalla. También puede introducir un LPRINT si desea tener la visión completa por escrito.

140: Ahora se utiliza el tercer byte (T+2) para hallar la dirección inicial de la siguiente línea BASIC

150: y se puede procesar la línea siguiente.

Una vez analizado todo este embrollo, ya no presenta gran dificultad escribir una rutina para poder cambiar los números de línea. Para ello sólo hace falta buscar los bytes cero y uno de cada línea de instrucción y depositar el nuevo número de línea deseado:

```
0 REM ADR137.BEC
1 REM *****
2 REM *
3 REM * CAMBIO DEL NUMERO DE LINEA *
4 REM * DE BASIC.
5 REM *
6 REM *****
10 A=25
20 B=7.5
30 C=17
40 F=A/B
50 G=B*C
60 H=C-A
70 PRINT F,G,H
1000 ZN=500:SW=25:Z=PEEK(136)+PEEK(137)*256
30000 HI=INT(ZN/256):POKE Z,ZN-HI*256:POKE
Z+1,HI:ZN=ZN+SW:Z=Z+PEEK(Z+2):GOTO 30000
```


10 a 70: Contiene el programa conejillo de indias.

1000: ZN toma el primer número de línea y SW la longitud de paso de la numeración. Z calcula el puntero que apunta hacia la primera línea BASIC.

30000: En esta línea se efectúa la conversión propiamente dicha. HI calcula de ZN la parte del byte HI. Después se coloca con POKE el resto del byte LO ($ZN - HI * 256$) en la dirección Z y en la dirección Z+1 el valor del byte HI. El próximo paso consiste en incrementar el número de línea con el valor de la longitud de paso SW, obteniendo así el próximo número de línea. Ahora se cambia el puntero Z por el valor del offset (PEEK(Y+2)). Finalmente se realiza el salto hacia 30000 para tratar la siguiente línea BASIC.

El programa finaliza con un mensaje de error. Cuando se haya efectuado el último cambio de número que corresponde a la línea 30000, GOTO ya no encontrará ninguna línea. Si escribe ahora un LIST sin variar los valores indicados, verá que la última línea tiene ahora el número 850.

Al definir los valores de ZN y SW debe tener la precaución de no sobrepasar el máximo número de línea admitido (32767). Al confeccionar una utilidad renumber es aconsejable comprobar esta condición, porque no siempre se puede apreciar hasta qué número de línea llegará el programa renumerado.

Con el cambio de números de línea también podrá impedir que se realice un listado de su programa. Una vez escrito en limpio el programa BASIC y almacenado en la memoria en forma token, se asigna un mismo número a todas las líneas. Este proceso ocurre según el mismo principio que el anterior cambio de los números de línea.

Una vez escrito el programa y registradas en orden correcto las líneas BASIC en la memoria, el ordenador ya no tiene en cuenta los números de línea. Incluso si todas las líneas BASIC tienen el mismo número, el programa continúa ofreciendo resultados correctos. También podrá grabarse y cargarse con normalidad utilizando SAVE y LOAD. Una vez almacenado en la memoria también podrá emitirse su listado en la pantalla. Sólo si se almacena con LIST en un periférico y se vuelve a cargar después, lo único que queda de todo el programa es la última línea.

El siguiente programa iguala todas las líneas:

```
0 REM ADR137X.BEC
1 REM *****
2 REM *
3 REM * DESTRUIR NUMEROS DE LINEA *
4 REM *
5 REM *****
10 A=25
20 B=7.5
30 C=17
40 F=A/B
50 G=B*C
60 H=C-A
70 PRINT F,G,H
1000 Z=PEEK(136)+PEEK(137)*256
30000 POKE Z,244:POKE Z+1,1:Z=Z+PEEK(Z+2):GOTO 30000
```

10 a 70: Contienen el programa postizo.

1000: Determina el puntero STMTAB.

30000: Cambia todos los números de línea a 500 ($244 + 1 \cdot 256$). Naturalmente, usted puede asignarles cualquier valor admitido como tal, por ejemplo incluso el número 0. Si se asigna un número que sobrepase el valor permitido, por ejemplo $LO=255$ y $HI=255$, la renumeración se efectúa sin ningún error, despidiéndose con ello el programa. Esta es una forma exclusiva de borrar un programa.

Puntero hacia la instrucción BASIC que se está ejecutando. Es utilizado para acceder a los token de una línea BASIC que se está procesando. Cuando BASIC está esperando que se efectúe una entrada, este puntero apunta hacia la línea de entrada directa (32768), abriendo así la posibilidad de proteger el programa de manera especial:

```

0 REM ADR138.BEC
1 REM *****
2 REM *                               *
3 REM * PROTECCION DE LIST CON RUN *
4 REM *                               *
5 REM *****
32766 POKE PEEK(138)+PEEK(139)*256+2,0:SAVE "D:NOMBRE
FICHERO.EXT":NEW
32767 REM SE ARRANCA CON GOTO 32766

```

32766: Añada esta instrucción como última línea a su programa y grábelo con el comando GOTO 32766. Sin embargo, antes de almacenar el programa con SAVE, puede evitar el salto hacia atrás a partir de la línea de entrada directa con ayuda de STMCUR. Entonces se almacena el programa en forma de token, o sea que se guardan todos los punteros -incluso STMCUR- con su valor actual. Después se borra el programa con NEW.

Esta protección también funciona con el cassette. Sustituya simplemente por "SAVE"C:". ¡Pero cuidado! Antes de crear una versión protegida de su programa según este método, es aconsejable asegurarse una versión no protegida del mismo, ya que se borra el programa fuente almacenado en la memoria del ordenador.

Puesto que el programa ha sido almacenado con SAVE, sólo podrá cargarse con LOAD. Después de la entrada en modo directo de LOAD"... " ya no se podrá efectuar ninguna otra

entrada directa, o sea ni LIST ni tampoco RUN. Sin embargo, los programas grabados con SAVE pueden ser cargados con RUN"D:nombre de fichero.ext" o bien RUN"C": y ejecutados inmediatamente. Esto significa que el programa puede funcionar igual, pero el usuario no puede hacer ningún listado del mismo ya que se está ejecutando.

Sin embargo, para conseguir una protección eficaz debe cumplirse la condición de que el usuario no tenga nunca la oportunidad de escribir un comando BASIC, LIST o cualquier otro. Para ello hay que conseguir que el programa sea absolutamente impermeable. No puede contener ningún fallo que provoque una interrupción del mismo con un mensaje de ERROR, porque entonces será posible escribir cualquier comando.

En STOPLN (186,187 = \$BA,\$BB) le mostraré la manera de asegurar un programa de manera absolutamente infalible contra las interrupciones por ERROR.

Además hay que bloquear RESET y BREAK para evitar que el usuario interrumpa la ejecución del programa. La manera de conseguir esto ya se ha explicado anteriormente. De hecho, si usted protege su programa según el método antes presentado, no es necesario tomar ninguna medida adicional. Sin embargo, y para prevenir cualquier eventualidad, puede cambiar además algún que otro puntero y, por ejemplo, sustituir todas las variables por RETURN.

Con todo esto usted ya dispone de una buena protección del programa. Al menos para uso casero. Aún y así puede copiarse un programa protegido de esta forma, y un profesional no tendrá que romperse demasiado la cabeza para invalidar todas estas graciosas medidas de seguridad. La verdadera protección del software se efectúa en otros campos de batalla.

Pero ya que estamos ocupados en juegos infantiles, le presentamos otro método que sirve para transportar mensajes secretos llamado 'programa BASIG como buzón muerto'.

Para ello necesitará un programa lo más inofensivo posible, introduciendo en algún lugar una línea REM (o también varias) que contenga su mensaje secreto:

```
0 REM ADR138S.BEC
1 REM *****
2 REM * *
3 REM * MENSAJE SECRETO PALABRA *
4 REM * CLAVE *
5 REM *****
100 REM PROGRAMA
200 REM PROGRAMA
300 REM PROGRAMA
400 REM GUDRUN, TE QUIERO
500 REM PROGRAMA
600 REM PROGRAMA
700 REM PROGRAMA
800 PRINT "PARA COMENZAR UN PARTIDO NUEVO,"
810 PRINT :PRINT "POR FAVOR PULSE<START>--"
820 IF PEEK(53279)=6 THEN RUN
830 IF PEEK(53279)=1 THEN GRAPHICS 0:GOTO 1000
840 GOTO 820
1000 PRINT CHR$(125):PRINT :PRINT "DESPUES DE INTRODUCIR LA
PALABRA CLAVE"
1010 PRINT :PRINT "SE GRABA EN EL DISKETTE UNA VERSION"
1020 PRINT :PRINT "DESPROTEGIDA DEL PROGRAMA"
1030 PRINT :PRINT :PRINT "EL PROGRAMA SE CARGA CON"
1040 PRINT "ENTRE";CHR$(34);"D:CIA.KGB";CHR$(34)
1050 DIM CW$(5)
1060 TRAP 1060
1070 INPUT CW$
1080 IF CW$="QRYXT" THEN LIST "D:CIA.KGB",0,1999
1090 GOTO 1060
20000 POKE PEEK(138)+PEEK(139)*256+2,0:SAVE
"D:BOTSCHFT.MAD":NEW
```

100 a 700: Representan el programa inocente -quizás sea un juego- que lleva el mensaje discreto en la línea 400.

800 y 810: El programa finaliza con la opción de comenzar otra partida pulsando START.

820: Al pulsar la tecla START, el programa se inicia nuevamente con RUN.

830: Solamente el receptor autorizado sabe que no debe pulsar START ni desconectar el ordenador al llegar a este punto, sino pulsar OPTION y SELECT simultáneamente, saltando de esta forma a la línea 1000.

840: Finaliza el bucle de lectura.

1000 a 1050: Aquí comienza la segunda rutina de protección, pidiendo la entrada de una palabra clave. Realmente, este programa se limita a comprobar sólo la contraseña entrada. Como es natural, en este punto pueden preverse con un poco de fantasía, todo un conjunto de medidas de protección adicionales. Se podría borrar todo el programa, por ejemplo, al detectar una contraseña errónea (NEW).

1060: Hay que proteger la entrada contra errores de entrada (quizás premeditados).

1080: En caso de acertar la contraseña correcta, se graba el programa con LIST en el diskette, sin almacenar tampoco la última línea que contiene la rutina de protección. Por supuesto también funciona de igual manera para el cassette.

2000: Aquí se encuentra la protección LIST/LOAD. Usted tendrá que grabar, pues, este programa haciendo GOTO 20000 antes de enviarlo a su amada con el mensaje destinado tan sólo a sus ojos.

Puntero hacia el primer byte de la tabla de las variables de campo (strings y arrays). Los arrays se almacenan con el formato de 6 bytes-BCD. ¿Esto quiere decir que al DIMensionar una variable con (9,9), se ocupan pues 10 por 10 por 6, o sea 600 bytes! A diferencia de ello, los strings sólo ocupan un byte por cada elemento, o sea DIMG\$(20) ocupa 20 bytes.

A menudo puede ahorrarse mucha memoria, definiendo como strings los números que sólo deben mostrarse con PRINT, en vez de hacerlo como valores numéricos. El siguiente programa le muestra un ejemplo:

```

0 REM ADR140.BEC
1 REM *****
2 REM *                               *
3 REM *  USO DE STRINGS EN LUGAR    *
4 REM *  EN LUGAR DE ARRAYS        *
5 REM *                               *
6 REM *****
10 DIM A(10),Z$(20)
20 A(0)=9:A(1)=11:A(2)=1984:A(3)=140
30 Z$="09111984ADR140"
40 ? A(0);".";A(1);".";A(2);"!";A(3)
50 ? :? Z$(1,2);".";Z$(3,4);".";Z$(5,8);": ";Z$(9,14)

```

```

*****
* ¡Atención! Línea 30 contiene pseudo-caracteres gráficos. *
*****

```

10: Muchos programadores Inexpertos DIMensionan además arrays y strings como provisiones.

20: A(0) a A(3) toman aquí números que sirven para indicar una fecha o un número de identificación. La memoria efectiva necesaria para ello es de 4 por 6 = 24 bytes.

30: Todos los caracteres necesarios se asignan a un string. La memoria necesaria es de catorce bytes, a pesar de que además contenga información adicional.

40: La fecha y el número de identificación se muestran con PRINT.

50: Aquí se consigue la misma indicación con partes del string. Un valor numérico no puede estar precedido de 0. La indicación del día 09 (en vez de 9 solamente) es válida como string y, tal como indica el número de identificación, se pueden mezclar fácilmente cifras y letras. Puesto que los strings también pueden convertirse hasta cierto punto en valores numéricos, utilizando el comando VAL del BASIC, también pueden efectuarse cálculos con los valores almacenados de esta forma.

Echemos ahora un vistazo a la tabla de las variables de campo:

```
0 REM ADR140A.BEC
1 REM *****
2 REM * *
3 REM * USO DE STRINGS EN LUGAR *
4 REM * DE ARRAYS *
5 REM * *
6 REM *****
10 DIM A(9),Z$(20)
20 REM FOR J=0 TO 9:A:A(J)=0:NEXT J
30 REM Z$=" "
40 A(0)=9:A(1)=11:A(2)=1984:A(3)=140
50 Z$="0911184 ADR140"
60 PRINT A(0);".";A(1);".";A(2);".";A(3)
70 PRINT :PRINT Z$(1,2);".";Z$(3,4);".";Z$(5,8);".";Z$(9,14)
100 PRINT CHR$(125)
110 SP=PEEK(140)+PEEK(141)*256
```

```

120 FOR J=0 TO 9
130 FOR I=0 TO 5
140 PRINT PEEK(SP+J*6+I);" ";
150 NEXT I
160 PRINT
170 NEXT J
200 FOR J=60 TO 79
210 PRINT PEEK(SP+J);" ";
220 NEXT J
230 PRINT
240 FOR J=60 TO 73
250 PRINT CHR$(PEEK(SP+J));
260 NEXT J

```

10 a 70: Contienen el programa que ya hemos explicado antes.

100: Limpia la pantalla.

110: Se calcula la dirección inicial de la tabla de strings y arrays con ayuda del puntero STARP.

120: Lee los bytes de las diez variables A.

130:: Lee los seis bytes de cada variable A.

140: Muestra con PRINT los bytes de una variable A separados por espacios intermedios.

160: Empezar con la siguiente variable A en una nueva línea, a fin de que los seis bytes se encuentren siempre uno debajo de otro.

200: Puesto que el array A ocupa 60 bytes, el string Z tiene valores (ATASCII) de 60 a 73.

210: Efectúa los PEEK y los PRINT de estos valores.

Ejecute el programa con RUN y verá que la tabla se ha convertido en un infierno de variables que han sido DIMENSIONADAS, pero que todavía no han recibido ningún valor. Esto puede conducir a errores fatales en los arrays.

Borre el REM de las líneas 20 y 30, sustituya el 73 por 79 en línea 240 y vuelva a ejecutar el programa. La tabla tiene ahora un aspecto limpio y pulido. Los elementos del array a los que todavía no se habían asignado valores se han rellenado con ceros. En la línea 30 se había llenado el string con una secuencia de espacios y corazones. Usted puede ver pues, que los elementos de un string son depositados con formato ATASCII, siendo 32 el de los espacios y 0 el de los corazones.

Los valores de arrays y string también pueden colocarse en la tabla directamente con POKE, siendo necesario saber en qué lugar se encuentra la variable en cuestión. Los elementos del string se anotan pues en formato ATASCII. Los valores numéricos se registran en formato de 6 bytes BCD. BCD (binary coded decimal) codifica cada una de las cifras de un valor numérico en forma binaria. Esto requiere mucha memoria y es más complicado a partir de las operaciones de cálculo, empleando pues más tiempo en ello, pero tiene la ventaja de ser mucho más exacto que el método del número entero.

El byte 0 de una constante BCD toma en bit 7 el signo (activado -, no activado +) y en los bits 6 a 0 el exponente de la base 100. Hay un exponente positivo cuando los bits 6 a 0 contienen un valor decimal igual o mayor que 64, o sea que el exponente tiene la misma magnitud que la diferencia entre dicho valor y 64. Así pues, el valor 64 indica que el exponente vale 0 ($100 \text{ elevado a } 0 = 1$, el número se encuentra entre 0 y 99); 65 indica que el exponente vale 1 (número entre 0 y 9999), etc. Los valores menores que 64 indican los correspondientes valores negativos, o sea las cifras decimales.

A los cinco bytes restantes se les asignan las cifras que componen el número. Para ello se anota cada cifra codificada de forma binaria en un nibble. Si la variable tiene por ejemplo el valor -12, el byte 0 tendrá el valor 192 (128+64) y el primer byte recoge el 12. El nibble superior toma el 1 en su bit inferior (1 decimal = 0001 binario); el nibble inferior toma el 2 (2 decimal = 0010 binario).

Si el número a anotar es 78, el byte 0 será 64, el 7 es colocado como 0111 en el nibble superior (equivale a 7 en notación binaria) y el 8 como 1000 en el nibble inferior:

NUMEROS DECIMALES CODIFICADOS EN BINARIO:

Cifra decimal	7				8			
Bits	0	1	1	1	1	0	0	0
Número del bit	7	6	5	4	3	2	1	0
Valor posicional	128	64	32	16	8	4	2	1

En el siguiente programa ejemplo se define el valor de un elemento del array directamente en la tabla. El número toma la sucesión de cifras 1234567890 y el punto decimal es situado por el byte del exponente entre las posiciones sexta y séptima. El 0 de la cuarta cifra decimal no aparece en pantalla:

```

0 REM ADR141.BEC
1 REM *****
2 REM * *
3 REM * ALTERAR VARIABLES DIM *
4 REM * *
5 REM *****
10 DIM Z(9)
20 FOR J=0 TO 9:Z(J)=0:NEXT J
30 SP=PEEK(140)+PEEK(141)*256
40 POKE SP,66:REM =100^2
50 POKE SP+1,18:REM =1/2
60 POKE SP+2,52:REM =3/4
70 POKE SP+3,86:REM =5/6
80 POKE SP+4,120:REM =7/8
90 POKE SP+5,144:REM =9/0
100 PRINT Z(0):PRINT :PRINT
110 FOR J=0 TO 5
120 PRINT PEEK(SP+J);" ";
130 NEXT J

```

10: El tamaño del DIMensionado es arbitrario.

20: Se asigna el valor 0 a todos los elementos del array, medida que debería convertirse en rutinaria después de cada DIMensionado.

30: Se determina la dirección hacia la cual apunta STARP.

40: El byte cero recoge el exponente y el signo.

50: El primer byte almacena las cifras 1 y 2 en código binario (0001/0010). De esta forma el byte toma valor decimal 18.

60: El segundo byte recoge las cifras 3 y 4 (0011/0100 = 52).

70: El tercero almacena el 5 y el 6 (0101/0110 = 86).

80: El cuarto byte recoge las cifras 7 y 8 (0111/1000 = 120).

90: El quinto y último byte se hace cargo de las cifras 9 y 0 (1001/0000 = 144).

100: Muestra el número decimal.

110 a 130: Muestran el sexto byte de datos de la correspondiente constante BCD.

142,143 \$8E,\$8F RUNSTK

Dirección del stack de runtime (pila para tiempo de ejecución); contiene notas sobre todas las instrucciones GOSUB (cuatro bytes para cada uno) y FOR-NEXT (dieciséis bytes para cada uno) que se están procesando en este momento.

Una inscripción GOSUB contiene en el byte cero un 0 para indicar "GOSUB", en byte 1 y 2 el número de línea que ha llamado al GOSUB, repartido como valor entero en LO y HI, y el último byte contiene el offset que se encuentra en la línea para que RETURN pueda volver y ejecutar la siguiente instrucción.

Una inscripción FOR-NEXT recoge en los bytes 0 a 5 el valor final de las variables contadoras, o sea el valor indicado detrás de T0 en forma de constante BCD; los bytes 6 a 11 contienen la indicación STEP, igualmente en forma BCD; los bytes 12 y 13 contienen el número de línea con el comando FOR y el último, el byte 15, contiene el offset de línea de la instrucción FOR.

Mediante RUNSTK se marca al mismo tiempo el final de la tabla de las variables de campo.

144,145 \$90,\$91 MEMTOP

Apunta hacia el final de la memoria ocupada por un programa BASIC. El comando FRE(0) calcula la memoria que queda disponible entre MEMTOP y el principio del display list y de la memoria de pantalla, restando para ello los respectivos punteros SDLSTL y MEMTOP.

Si se pretende limitar espacio de memoria, hay que registrarlo aquí y en RAMTOP (106 = \$6A).

186,187 \$BA,\$BB STOPLN

Almacena el número de línea donde se ha producido una interrupción del programa provocada por ERROR-, STOP, TRAP o pulsación de la tecla BREAK.

Con este registro se puede impedir cualquier interrupción provocada por un error del programa, por ejemplo en lo que a protección de software se refiere, o porque es la forma más simple de saltarse una interrupción del programa a causa de valores numéricos incorrectos.

En el programa siguiente se DIMensiona una variable doblemente indexada cuyo elemento se incrementa a continuación en valores aleatorios. Puesto que estos valores aleatorios pueden ser mayores que la dimensión prevista del array, se emplea el truco TRAP:

```

0 REM ADR186.BEC
1 REM *****
2 REM *                               *
3 REM *  PROTECCION DE ERROR          *
4 REM *                               *
5 REM *****
9 TRAP 9:F=F+1:GOTO PEEK(186)+PEEK(187)*256+10
10 REM DIRECCION FINAL DE BIFURCACION TRAP EN EL PRIMER PASO
100 DIM M(6,6):FOR I=0 TO 6:FOR J=0 TO 6:M(J,I)=0:NEXT J:NEXT
I
110 X=INT(RND(0)*6)
120 Y=INT(RND(0)*6)
130 AL=4-X:AR=4+X
140 BL=4-Y:BR=4+Y
150 M(AL,BL)=M(AL,BL)+1
160 M(AR,BL)=M(AR,BL)+1
170 M(AR,BR)=M(AR,BR)+1
180 M(AL,BR)=M(AL,BR)+1
190 Z=Z+1:IF Z<50 THEN 110
200 FOR J=0 TO 6
210 FOR I=0 TO 6:PRINT M(J,I);" ";:NEXT I:PRINT
220 NEXT J
230 PRINT :PRINT :PRINT F

```


9: Se activa el TRAP y se bifurca hacia él, es decir que cada vez que el TRAP entra en acción, el programa llega a esta línea en la que el TRAP se renueva. La instrucción GOTO que le sigue lee en el puntero STOPLN la línea donde se ha efectuado el TRAP, suma 10 al número de ésta y después conduce el programa hacia la línea resultante.

Sin embargo, este proceso sólo funciona si el programa entero está numerado de 10 en 10 y si en la última línea no es posible que se produzca ningún error, ya que en caso contrario la misma instrucción GOTO generaría un error y el programa no podría salir de dicha línea.

Si queremos proteger software, una línea de este tipo proporciona la absoluta seguridad de que el usuario del programa sea incapaz de abrirlo utilizando alguna entrada errónea. Tampoco tiene importancia que el programa pueda continuar en algún lugar después del TRAP, porque éste volvería a interceptarlo de todos modos...

También podría admitirse la forma siguiente:

```
100 TRAP 20000
20000 NEW:LOAD"D:nombre de fichero.ext"
```

En este programa ejemplo la variable F cuenta por curiosidad la cantidad de mensajes de error que se han podido evitar utilizando el truco del TRAP.

10: Cuando el programa se está ejecutando por primera vez, el puntero STOPLN tiene valor 0, procesándose un GOTO 10 en la línea 9. Esto exige que el programa contenga la línea 10, aunque sólo sea una línea REM.

100: La variable del array M toma la DIMensión de siete por siete y se asigna el valor 0 a todos sus elementos.

110 y 120: X e Y obtienen valores aleatorios entre 0 y 5.

130 y 140: Con ayuda de X e Y, las variables AL, AR, BL y BR toman algunos valores que no sirven como números de identificación para el array dimensionado. De utilizarlos como tales en las líneas

150 a 180: el resultado será siempre el mensaje de error "cantidad ilegal". Sería muy complicado comprobar los valores ilegales de X e Y mediante condiciones IF, especialmente en aquellos casos en que sólo una de las variables sobrepasara el límite admitido, puesto que se incrementa incluso el valor legal por el que se puede acceder a un determinado elemento del array.

Con la estructura del TRAP de la línea 9 se facilita la solución del problema. Si se intenta llamar a un elemento ilegal del array, el correspondiente mensaje de error conduce el programa a la línea siguiente. Esto significa que, independientemente del número aleatorio que le permite acceder a un determinado elemento de la variable M doblemente indexada, M también considerará el caso en que el número aleatorio no sirva, con lo que el programa seguirá a pesar de ello.

190: El programa de prueba se procesa cincuenta veces.

200 a 220: A continuación se muestra la variable de campo, y en la línea

230: la variable F nos informa sobre la cantidad de veces que el número aleatorio ha fallado.

195 \$C3 ERRSAVE

Contiene el número del error que ha conducido a un STOP o TRAP.

201 \$C9 PTABW

Determina la cantidad de columnas entre tabulaciones, ordenadas en una instrucción PRINT mediante una coma, es decir la distancia entre el último carácter del PRINT precedente y el primero del siguiente. Esta función no depende de la tecla TAB.

El valor default es 10. El mínimo valor admitido es 3. Se suma el valor 2 al valor que ha sido POKEado después de PTABW; o sea POKE 201,1 produce una longitud de salto tabulador de tres columnas. El valor máximo admitido es 255, que corresponde a una distancia de 257 columnas.

Si se escribe un 0 después de PTABW, el sistema se "cuelga" con la primera coma de una instrucción PRINT y sólo puede recuperarse con RESET.

212-241 \$D4-\$F1 diverse

Registro que se utiliza para operaciones de floating point (coma flotante).

242 \$F2 CIX

Índice de caracteres. Offset del buffer de texto de entrada; apunta hacia el INBUFF.

243,244 \$F3,\$F4 INBUFF

Apunta hacia el buffer de entrada de texto, es decir el buffer de entrada para líneas de programa del usuario.

245-250 \$F5-\$FA ZTEMP1-3

Registro Intermedio.

251 \$FB RADFLG

Flag (Identificación) para RADianes (unidad de arcos) o DEG (grados). El valor default 0 hace que todas las funciones trigonométricas se procesen en radianes (base 100). Si se sustituye por 6 o se da la instrucción BASIC DEG, se cambia a grados (base 360).

256-511 \$100-\$1FF page 1

Esta es la zona destinada al stack (pila) del OS, DOS y BASIC. Las instrucciones en lenguaje máquina JSR y PHA y las interrupciones provocan que los datos se escriban en la página 1; RTS, PLA y RTI leen datos de la página 1.

En powerup o reset el puntero del stack apunta hacia la dirección 511. Se escribe en el stack hasta bajar a 256. Cuando se produce su overflow, éste vuelve al 511.

512,513 \$200,\$201 VDSLST

Vector para el NMI (non-maskable interrupt) display list interrupt (DLI). Aquí se deposita la dirección hacia la cual se debe saltar durante un DLI.

Los DLI se utilizan para procesar otras tareas del programa durante los microsegundos del blank horizontal, como por ejemplo crear una melodía. Ello puede dar la impresión de sucesos simultáneos.

En el DLI también se pueden modificar los valores de los registros de colores, el modo gráfico, etc., lo que permite representar un color diferente en cada línea del modo, de manera que los 256 tonos de colores posibles puedan aparecer en la pantalla al mismo tiempo.

El OS no utiliza ningún DLI. Es necesario que el usuario los defina, escriba y vectorice. VDSLST se inicializa de tal forma que apunta hacia 59315 (\$E7B3). Para definir un DLI, la 54286 (\$D40E) debe ponerse previamente a 192 para que ANTIC pueda reconocer el request, y a continuación POKEar a 512 (LO) y 513(HI) la dirección donde comienza la rutina en lenguaje máquina a procesar durante el DLI. En el display list debe activarse el bit 7 (+128) en la instrucción de la línea del modo que precede a un DLI. Según el modo gráfico, el DLI dispone de 14 a 61 ciclos de máquina. En primer lugar es necesario empujar los 6502 registros hacia el stack y finalizar el DLI con un RTI. Puesto que el DLI se procesa en lenguaje máquina, los cambios deseados pueden POKEarse directamente en los registros del hardware. Los cambios de los registros de sombra accesibles a través de BASIC sólo son leídos por el VBLANK. Por esta razón, los cambios de valores por ejemplo en los registros de colores, se procesan ópticamente a gran velocidad, pero éstos inciden sobre la totalidad del contenido de la pantalla. Sin embargo, un DLI permite por ejemplo poner el registro de colores a 710 (fondo en el modo GRAPHICS 0) o 140 (azul claro = azul celeste) y a 182 (verde claro) para el resto de la ventana gráfica después de algunas líneas de modo.

```

0 REM ADR512.BEC
1 REM *****
2 REM * *
3 REM * DISPLAY LIST INTERRUPT DLI *
4 REM * *
5 REM *****
10 POKE 710,140:POKE 712,0:POKE 709,2
20 X=PEEK(560)+PEEK(561)*256
30 P=1536:FOR DLI=P TO P+10:READ B:POKE DLI,B:NEXT DLI
40 DATA 72,169,182,141,10,212,141,24,208,104,64
50 POKE 512,0:POKE 513,6
60 POKE 54286,192
70 POKE X+6+J,130
80 FOR I=0 TO 300:NEXT I
90 POKE X+6+J,2:J=J+1:IF J<2: THEN 70
100 REM GOTO 100

```

10: Se POKEan los valores de los colores. El registro 172 determina el borde (0 = negro) en GRAPHICS 0, y el 709 la claridad de los caracteres.

20: Calcula el puntero SDLSTL que apunta hacia el display list.

30: P es la dirección inicial de la rutina en lenguaje máquina, que corresponde al principio de la página 6 ($6 \cdot 256 = 1536$). En este lugar se encuentra por debajo del LOMEM una pequeña zona del RAM de aplicaciones protegida. El bucle FOR-NEXT lee en las celdas de memoria situadas detrás de P los bytes de datos que forman la rutina en lenguaje máquina.

40: Es la rutina en lenguaje máquina. Además de este método para colocar en la memoria un programa en lenguaje máquina en forma de bytes de datos, existe la posibilidad de colocarlo en un string. Los bytes de datos de esta línea podrían convertirse en strings mediante la instrucción:

```
FOR J=0 TO 10:READ A:B$(J,J)=CHR$(A):NEXT J
```

De esta forma, la rutina en lenguaje máquina se salvaría en la tabla de las variables de campo y se desplazaría junto a la memoria en sus desplazamientos internos, lo que le evita al usuario el tener que preocuparse en colocarla en algún lugar protegido contra escrituras. El comando BASIC ADR(B\$) puede fijar la dirección inicial de un string en la tabla de strings y arrays, y el comando BASIC USR pone el ordenador a trabajar: USR(ADR(B\$)).

50: El vector DLI se coloca en la página 6 (LO=0, HI=6).

60: Los bits 7 y 6 deben estar activados (decimal 192) en el NMIEM de ANTIC ($54286 = \$D40E$) para permitir el DLI.

70: En el display list, el byte del comando ANTIC cambia de 2 (correspondiente a una línea de GRAPHICS 0) a 130 (2+128 correspondiente a una línea con DLI de GRAPHICS 0).

80: Un pequeño descanso para la vista del usuario.

90: El comando ANTIC vuelve a convertirse en 2. A continuación se incrementa J en 1, de manera que en la próxima pasada se añada el DLI en la línea siguiente. Ello produce el efecto de que la parte superior de la ventana gráfica (cielo) se extienda lentamente, línea por línea del modo, hacia abajo, como si despegase un avión y se elevase hacia el cielo.

Si no le gustan los colores (respetamos todos los gustos) puede graduar el color del cielo con POKE 710,n; el del borde con POKE 712,n y la claridad de los caracteres con POKE 709,m. Para n y m puede elegir cualquier valor par entre 0 y 254, aunque para m sólo son interesantes los valores pares entre 0 y 14, puesto que únicamente afectan a la claridad.

Si desea cambiar el color después del DLI, o sea en la parte inferior de la ventana gráfica, deberá modificar el correspondiente byte de datos de la rutina en lenguaje máquina. Este corresponde al tercer valor numérico de la línea 40. También en este caso sólo se admiten valores pares.

Puesto que sólo existe esta única dirección del vector de DLI, si se quieren ejecutar varios DLI, el vector debe ser modificado por la propia rutina DLI que le precede. Además se aconseja suprimir el clic del teclado (registro 731 = 2DB) o bien impedir las entradas por teclado, puesto que pueden producirse interferencias con el DLI, provocando las pertinentes distorsiones.

514,515 \$202,\$203 VPRCED

Vector de la rutina IQR para la interfase en serie.

516,517 \$204,\$205 VINTER

Vector de la rutina IQR de la interfase en serie.

518,519 \$206,\$207 VBREAK

Vector para el comando 6502 BRK (\$00). (Esto no tiene nada que ver con la tecla BREAK.)

520,521 \$208,\$209 VKEYBD

Vector para la interrupción de teclado del POKEY. Se utiliza para provocar una interrupción a través de cualquier tecla del keyboard (además de BREAK). Inicializa a 65470 (rutina IQR de teclado del OS).

522,523 \$20A,\$20B VSERIN

Vector del POKEY hacia la rutina encargada de recibir datos en serie.

524,525 \$20C,\$20D VSEROR

Vector del POKEY hacia la rutina encargada de emitir datos en serie.

526,527 \$20E,\$20F VSEROC

Vector del POKEY para finalizar una transferencia de datos en serie.

528-533 \$210-\$215 VTIMR1,2,4

Vectores de interrupción para los timer 1, 2 y 4 (AUDF1, 2, 4) del POKEY. Cuando el timer del POKEY ha llegado a 0, se produce un salto hacia la dirección que apunta al vector correspondiente.

534,535 \$216,\$217 VIMIRQ

Vector principal de interrupción IRQ.

536,537 \$218,\$219 CDTMV1

Timer 1 del sistema, que cada 1/50 segundo cuenta en sentido decreciente hasta 0. Una vez llegado a 0 se efectúa un salto hacia la dirección memorizada en el correspondiente vector CDTMA1 (550,551 = \$226,\$227).

No es aconsejable usar el timer 1, puesto que lo utiliza el OS para las rutinas I/O.

538-545 \$21A-\$221 CDTMV2-5

Igual que CDTMV1.

546,547

\$222,\$223

VVBLKI

Vector para una interrupción de VBLANK inmediata. Este puntero puede ser modificado para poder ejecutar rutinas en lenguaje máquina de una longitud aproximada de 3.500 tiempos de máquina durante el blank vertical.

548,549 \$224,\$225 VVBLKD

Puntero para una interrupción del VBLANK retardada, que puede llegar a tener una longitud de 20.000 ciclos de máquina.

550,551 \$226,\$227 CDTMA1

Dirección de salto para el timer 1 (CDTMV1).

552,553 \$228,\$229 CDTMA2

Dirección de salto para el timer 2 (CDTMV2).

554 \$22A CDTMF3

Flag (identificación) para el timer 3 (CDTMV3)

555 \$22B SRTIMR

Timer utilizado para el rebote del teclado. Provoca el retardo antes de activar la función de repetición del teclado.

Cada vez que se pulsa una tecla, este timer comienza a partir de 8. Si SRTIMR llega hasta 0 y la tecla todavía se mantiene pulsada, el valor de la tecla pulsada pasa a CH (764 = \$2FC) con una velocidad de repetición de 1/10 segundos.

556 \$22C CDTMF4

Flag para el timer 4 (CDTMV4).

557 \$22D INTEMP

Registro intermedio utilizado por la rutina SETVBL.

558 \$22E CDTMF5

Identificación (flag) para el timer (CDTMV5).

559 \$22F SDMCTL

Registro de sombra de 54272. Control DMA (direct memory access = acceso directo a la memoria) de ANTIC (vea también apéndice gráfico player-missile).

Bit 5 (32) permite que ANTIC recoja las instrucciones de display list

Bit 4 (16) resolución de una línea del player en el gráfico PM

Bit 3 (8) posibilita player DMA

BIT 2 (4) posibilita missile DMA

Bit 1 (2) *) anchura del display

Bit 0 (1) *) anchura del display

*) Los bits 1 y 0 concretan la anchura de la pantalla de televisión que será operada por ANTIC.

Display de la anchura de bit 1 activado, bit 0 activado (designado literalmente por "palyfield" o campo de juego, por su relación con gráfico player-missile). Un display ancho admite 48 caracteres por línea en GRAPHICS 0. Sin embargo, de esta forma se modifica solamente la propia representación, mientras que el editor continúa trabajando con 40 caracteres por línea. Esto significa que al listar un programa, por ejemplo, después de ampliar el display activando los dos bits inferiores, las líneas del mismo aparecerán desplazadas por el valor desviado. Así pues, al trabajar con una anchura de 48 caracteres, el editor empieza la impresión del primer carácter de la línea siguiente en la

la impresión del primer carácter de la línea siguiente en la posición 41, etc.

Bit 1 activado, bit 0 desactivado; da la anchura habitual de 40 caracteres.

Bit 1 desactivado, bit 0 activado; corresponde a un estrecho campo de juego de 32 caracteres. En este caso, el editor imprime en las primeras ocho posiciones de la línea siguiente los últimos caracteres de la línea anterior, y empieza en la novena posición con la segunda línea.

Bit 1 desactivado, bit 0 desactivado; no hay display.

Es posible desactivar completamente la salida a pantalla a través de ANTIC, colocando a 0 los bits 5, 1 y 0. Y puesto que con ello se descarga el sistema completo, algunos procesos se efectúan con mayor rapidez. Interrumpir la salida a pantalla es especialmente interesante cuando se trata de realizar cálculos complejos, pero un programa que se sirve de ello, debería informar previamente al usuario de que a continuación se apagará su pantalla por un momento, para que éste no crea que se trata de una distorsión del televisor.

La forma más fácil de hacerlo es PEEKeando en primer lugar el valor actual de la dirección 559 y guardarlo en una variable. Después POKEar 0 a 559. Para volver a activar la pantalla, escriba en SDMCTL el valor de las variables: POKE 559, variable, asegurándose de esta manera que después de la desconexión este registro volverá a tener su valor antiguo.

La desconexión de la pantalla también puede ser eficaz cuando se crea una imagen en un modo gráfico elevado, lo que puede tardar varios minutos. Si desconecta antes el display, salva el gráfico en la memoria de pantalla y además vuelve a conectar en 559, el gráfico completo aparecerá de golpe. Un método aún más elegante ha sido presentado anteriormente con el programa ejemplo PAGING.BEC (SAVMSC 88,89 = \$58,\$59).

560,561

\$230,231

SDLSTL

Registros de sombra de 54274 y 54275. Vector hacia el display list (DL). El DL se encuentra inmediatamente delante de la memoria de pantalla. Es un corto programa en lenguaje máquina que controla a ANTIC. Contiene instrucciones sobre el lugar de la memoria donde se encuentran los datos de la imagen y la forma de interpretarlos. Existen comandos ANTIC para cada modo gráfico, para el DLI y para el scroll vertical y horizontal, así como dos instrucciones de salto (vea apéndice display list).

Si desea echar un vistazo al display list puede probar este programa:

```

0 REM ADR560.BEC
1 REM *****
2 REM * *
3 REM * IMPRIMIR DISPLAY LIST *
4 REM * *
5 REM *****
10 DIM D(201)
20 PRINT CHR$(125)
30 PRINT :PRINT "ENTRE POR FAVOR MODALIDAD DE GRAFICOS"
40 PRINT :PRINT "VALORES DESDE 0 HASTA 11 (XL HASTA 15)"
50 PRINT :PRINT "MODALIDADES DE GRAFICO CON VENTANA":? :? "DE
TEXTO + 16"
60 PRINT :PRINT "Y PULSE <RETURN>"
70 PRINT :PRINT "-----"
100 INPUT G
110 GRAPHICS G
120 DL=PEEK(560)+PEEK(561)*256
130 FOR J=0 TO 201
140 D(J)=PEEK(DL+J)
150 Z=Z+1
160 IF D(J)=65 THEN POP :GOTO 180
170 NEXT J
180 J=Z:D(J)=PEEK(DL+J)
190 D(J+1)=PEEK(DL+J+1)
200 GRAPHICS 0
210 POKE 201,1:POKE 83,37
220 FOR J=0 TO Z+1
230 PRINT D(J),
240 NEXT J
250 GOTO 250

```

10: Se asignan los bytes de datos del DL a la variable de campo D. El DL más largo (GRAPHICS 8+16) tiene una longitud de 202 bytes.

20 a 70: Muestra el texto con las instrucciones.

100: Entradas para G entre 0 y 31 conducen a resultados lógicos. El programa no está protegido contra entradas

- 110: Se conecta el modo gráfico escogido por el usuario.
- 120: DL toma las direcciones iniciales del display list.
- 130: Se asigna el valor máximo al contador del bucle.
- 140: Un byte de datos del DL es PEEKeado y anotado en D(n).
- 150: Z cuenta los bytes leídos.
- 160: 65 (JVB, jump at vertical blank = salto hacia el blank vertical) es la última instrucción del DL al que le sigue solamente el destino del salto (dos bytes). Cuando se encuentre un 65 en la lectura, el programa se despide con un POP en el bucle FOR-NEXT y prosigue la ejecución en la línea
- 180: donde se lee el penúltimo byte y en línea
- 190: leerá el último byte del DL.
- 200: Se conecta GRAPHICS 0 para la salida de los datos de DL.
- 210: Se define la longitud de PRINT-TAB (PTABW) en 1, o sea en tres columnas, y se marca el margen derecho (RMARGN) en 37.
- 220 a 240: Imprimen el byte de datos del DL en la pantalla.
- 250: Evita que el programa finalice con READY. Con este formato, los datos de GRAPHICS 0 llenan la totalidad de la pantalla. De finalizar el programa con READY, se produciría el scroll de las dos líneas superiores, las cuales desaparecerían de la pantalla. Por esta razón usted mismo tendrá que finalizar el programa utilizando BREAK.

El significado de los diversos valores numéricos en DL se comenta en el apéndice display list.

562 \$232 SSKCTL

Registro de sombra de 53775. Control de la interfase en serie.

563 \$233 LCOUNT

Contador de bytes para rutinas de carga (loader).

564 \$234 LPENH

Registro de sombra de 54284. Posición horizontal del lápiz óptico.

565 \$235 LPENV

Registro de sombra de 54285. Posición vertical del lápiz óptico (lightpen). Los valores para las posiciones del lápiz óptico corresponden a los mismos que aparecen en el gráfico PM; se desvían de los valores de columnas y líneas que se dan habitualmente en los modos gráficos.

566,567 \$237 BRKKY

Vector para la interrupción a través de la tecla BREAK. Puede ser reactivado para bifurcarse hacia una rutina de aplicaciones programada y tratar la interrupción de la tecla BREAK para borrar, por ejemplo, el programa de la memoria o iniciar un nuevo proceso de carga al activar la tecla BREAK en el interior de una protección de software.

568,569 \$238,\$239 libre

570-575 ~~570-575~~ \$23A-\$23F ...

VARIABLES AUXILIARES INTERNAS UTILIZADAS PARA PROCESAR DATOS EN SERIE.

576-579 \$240-\$243 ...

VARIABLES AUXILIARES INTERNAS UTILIZADAS PARA EL BOOTING DEL DISKETTE.

580 \$244 COLDST

Identificación (flag) del arranque en frío. Cualquier valor excepto 0 indica que se está procesando la rutina de inicialización del powerup. Si COLDST tiene valor 0, RESET conduce a un arranque en caliente.

623 \$26F GPRIOR

Registro de sombra de 53275. Determina la prioridad de representación de los diversos elementos de la imagen en gráficos PM cuando éstos se superponen. Esto produce un efecto óptico de delante-atrás. Los player que se superponen pueden adquirir un tercer color y los cuatro proyectiles (missiles) pueden utilizarse como jugador (player) quinto.

(Abreviaciones: P=jugador, PF=campo de juego, BAK=fondo y borde)

Bit 7	(128)	*) modo GITA
Bit 6	(64)	*) modo GITA
Bit 5	(32)	P superpuestos adquieren el tercer color
Bit 4	(16)	P quinto en lugar de cuatro proyectiles
Bit 3	(8)	orden de prioridad: PF 0-1, P 0-3, PF 2-3, BAK
Bit 2	(4)	orden de prioridad: PF 0-3, P 0-3, BAK
Bit 1	(2)	orden de prioridad: P 0-1, PF 0-3, P 2-3, BAK
Bit 0	(1)	orden de prioridad: P 0-3, PF 0-3, BAK

*) Los bits 7 y 6 posibilitan el modo GITA 9, 10 y 11. Estos modos gráficos utilizan el mismo display list que GRAPHICS 8, cambiando simplemente la interpretación de los datos de pantalla. En lugar de representar en dos colores un bit por pixel, o sea 320 pixel por línea, en estos tres modos GITA se leen cuatro bits por pixel. Con ello se distinguen un máximo de dieciséis matices diferentes de colores. Para llegar a la misma cantidad de datos, cada pixel se representa cuatro veces más ancho, con lo que de esta forma se llena una línea del modo con 80 pixel.

Bit 7 desactivado, bit 6 activado; GRAPHICS 9 (los pixel tienen el mismo matiz, pero se admiten dieciséis gradaciones diferentes de claridad).

Bit 7 activado, bit 6 desactivado; GRAPHICS 10 (se admiten nueve matices de definición propia para los pixel, ¡disponiendo sólo de nueve registros de colores!).

Bit 7 activado, bit 6 activado; GRAPHICS 11 (los pixel pueden tener cualquiera de los dieciséis colores estándar, pero la claridad es la misma para todos).

Si el bit 5 está activado, se crea un nuevo color al superponerse los player 0 y 1 o los player 2 y 3. Los dos registros de colores correspondientes se enlazan mediante la operación lógica OR. Las superposiciones de otras combinaciones de player no dan un matiz nuevo. Si el bit 5 está desactivado, la zona de la pantalla ocupada por dos jugadores deviene negra.

Al activar el bit 4, la memoria de PM reservada para los cuatro proyectiles es tratada como un quinto jugador.

Puede ocurrir que se definan prioridades contradictorias en los bits 0 y 3. Si ello provoca un caso conflictivo, la zona de la pantalla afectada deviene negra.

624 \$270 PADDLO

Contiene el valor del paddle 0. En algunas ocasiones los paddles (raquetas) también se designan pots (potenciómetro, regulador giratorio).

625-631 \$271-\$277 PADDLE1-7

En los modelos XL ya no se utilizan más que los paddles 0 a 3.

632 \$278 STICK0

Contiene el valor del joystick 0. Existen nueve valores numéricos diferentes para este registro, modificando sólo los cuatro bits inferiores.

Bit 3 a 0 activado (0000 1111 = 15) joystick en posición de reposo.

Bit 3 desactivado (0000 0111 = 7) joystick a derecha

Bit 2 desactivado (0000 1011 = 11) joystick a izquierda

Bit 1 desactivado (0000 1101 = 13) joystick abajo

Bit 0 desactivado (0000 1110 = 14) joystick arriba

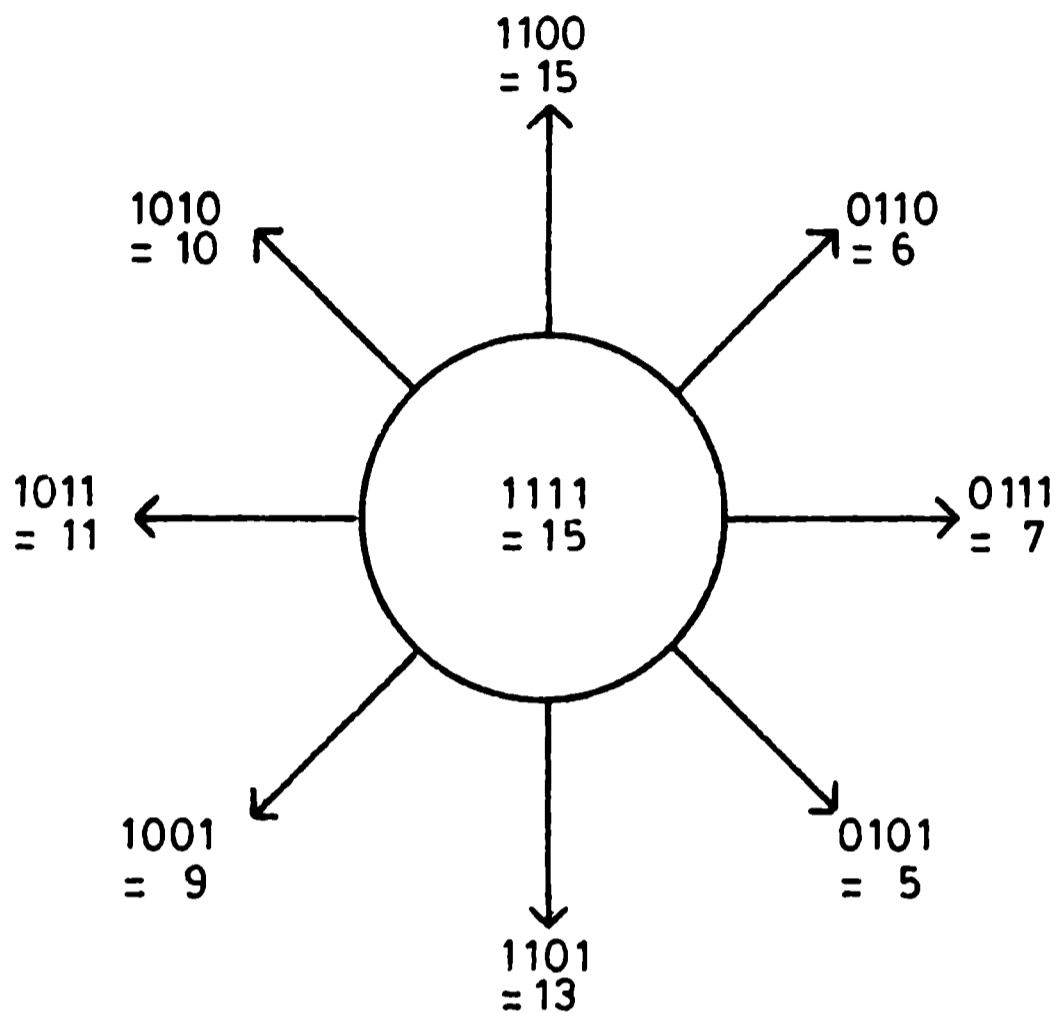
Bits 3 y 1 desactivados (0101 = 5) joystick abajo derecha

Bits 3 y 0 desactivados (0110 = 6) joystick arriba derecha

Bits 2 y 1 desactivados (1001 = 9) joystick abajo izquierda

Bits 2 y 0 desactivados (1010 = 10) joystick arriba izquierda

ILUSTRACIONES
posiciones - joystick



Posiciones del joystick
Configuraciones del bit y valores decimales

633-635

\$279-\$27B

STICK1-3

Igual que stick 0. En modelos XL sólo STICK0 y 1.

636 \$27C PTRIGO

Indica si el botón de fuego del paddle 0 está pulsado. Contiene un 0 si el botón está pulsado y un 1 en caso contrario.

637-643 \$27D-\$283 PTRIG1-7

Igual que PTRIGO. En los modelos XL sólo existen de 0 a 3.

644 \$284 STRIGO

Igual que PTRIGO, pero para el joystick.

645-647 \$285-\$287 STRIG1-3

Igual que STRIGO. Los XL sólo disponen del 0 y 1.

656 \$290 TXTROW

En el modo split-screen (gráfico con ventana de texto), la ventana gráfica es administrada por el motor del display ("S:"), mientras que el editor ("E:") controla la ventana de texto. Se utilizan IOCB separados y existen dos cursores independientes. Los datos gráficos de la ventana de texto se guardan, además, en una zona de la memoria bien definida y separada de la memoria de pantalla restante (vea apéndice memoria de pantalla).

TXTROW contiene la línea del cursor de la ventana de texto. Puesto que la ventana de texto tiene una altura de cuatro líneas, este registro sólo puede tomar valores de 0 a 3. Al ampliar la ventana de texto a cinco o más líneas modificando el display list, aparecen los mismos problemas de posiciones

de cursor ilegales que ya han sido comentados en GRAPHICS 71/2 relacionado con el registro DINDEX (87 = \$52).

657,658 \$291,292 TXTCOL

Columna del cursor de la ventana de texto. Contiene valores de 0 a 39. En todos los display lists estándar, el byte HI (658) tiene valor 0.

Los comandos BASIC POSITION, PLOT o LOCATE se refieren únicamente al cursor de la ventana gráfica, por lo que son ineficaces en la ventana de texto, donde sólo pueden ser sustituidos por los POKES correspondientes (vea apéndice memoria de pantalla).

659 \$293 TINDEX

Contiene el actual modo gráfico de la ventana de texto. Es el equivalente a DINDEX (87 = \$57) de la ventana de texto y su valor siempre es 0 si SWPFLG (123 = \$7B) es 0.

TINDEX es inicializado a 0. Sin embargo, el modo gráfico de la ventana de texto puede programarse libremente con la pertinente modificación del DL. Si se efectúa el cambio, hay que adaptar el valor de este registro.

660,661 \$294,\$295 TXTMSC

Dirección inicial de la memoria de pantalla correspondiente a la ventana de texto. Esta memoria de pantalla ocupa una determinada zona, independiente de la ocupada por la ventana gráfica. Es por esta razón que la ventana de texto puede aparecer y desaparecer de la pantalla, sin que por ello se pierdan los datos de la zona de memoria ocupada alternativamente por la ventana gráfica y la de texto.

TXTMSC muestra la celda de memoria que contiene los datos de pantalla correspondientes a la esquina superior izquierda de la ventana de texto. Es el equivalente a SAVMSC (88,89 = \$58,\$59) y puede ser manipulado de la forma allí descrita.

662-667 \$296-\$29B TXTOLD

Estos registros contienen los equivalentes a OLDROW (90 = \$5A), OLDCOL (91,92 = \$5B,\$5C), OLDCHR (93 = \$5D) y OLDADR (94,95 = \$5E,\$5F) que corresponden a los datos del cursor de la ventana de texto.

668-671 \$29C-\$29F -

Registro Intermedio.

672 \$2A0 DMASK

Máscara que sirve para definir en un byte de datos gráficos los bits pertenecientes a un pixel. Según el modo gráfico, se pueden agrupar ocho, cuatro, dos o un solo pixel en un byte. La máscara contiene un 0 por cada uno de los bits innecesarios para la representación del pixel actual y un 1 por los bits del pixel.

Según el modo gráfico utilizado, se activa la respectiva máscara:

11111111 GRAPHICS 0, 1, 2 8 bit/pixel = 1 pixel/byte

11110000

00001111 GRAPHICS 9,10,11 4 bit/pixel = 2 pixel/byte

11000000		
00110000		
00001100	GRAPHICS 3, 5, 7	
00000011	12,13,15	2 bit/pixel = 4 pixel/byte
10000000		
01000000		
a		
00000010	GRAPHICS 4, 6, 8	
00000001	14	1 bit/pixel = 8 pixel/byte

673 \$2A1 TMPLBT

Memoria intermedia utilizada por la máscara de bits.

674 \$2A2 ESCFLG

Identificación de escape. Su valor normal es 0 y se pone a 128 al pulsar la tecla ESCAPE. Después de emitir el siguiente carácter vuelve a ponerse inmediatamente a 0. Si se pretenden representar los caracteres de control ATASCII sin utilizar ESCAPE, DSPFLG (766 = \$2FE) puede ser activado con un valor distinto de 0.

675-689 \$2A3-\$2B1 TABMAP

Máscara utilizada para definir las paradas de tabulación.

La máscara TAB se refiere a la línea lógica, que comprende tres líneas físicas (GRAPHICS 0) de 40 columnas cada una, dando un total de 120 posiciones de columna. La máscara de TAB está definida por 15 bytes (15*8 bits = 120). Cada bit activado ocasiona una parada de tabulación que, mientras no se modifique, es válida para todas las líneas lógicas. Sin embargo, pueden ser diferentes para tres líneas físicas consecutivas.

La máscara tabuladora se define a través del teclado utilizando las teclas TAB-SET y TAB-CLR. El margen izquierdo de la pantalla LMARGN (82 = \$52) también hace las veces de parada tabuladora.

El valor default para todos los bytes de TABMAP es 1, efectuando paradas de tabulación en las columnas 7, 15, 23, etc. Los valores default se renuevan con RESET y con cada comando GRAGHICS y OPEN dirigido a "S:" o "E:". La máscara tabuladora también es efectiva en la ventana de texto.

Para cambiar esta máscara con POKE, hay que activar las paradas tabuladoras en los bits de diversos bytes, sumar sus valores posicionales y POKEar el resultado así obtenido.

Ejemplo de una máscara TAB:

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	etc.
00001000	01000010	00010000	10000100	00100001	00001000	etc.
= 8	= 66	= 16	=132	= 33	= 8	etc.

Para fijar una parada de tabulación cada cinco columnas, se deben colocar los valores decimales 8, 66, 16, 132, 33, 8, etc. en los registros 675 ff.

Un salto tabulador se origina al pulsar la tecla TAB y puede ser programado con PRINT "ESC" TAB.

Máscara que sirve para marcar el principio de las líneas lógicas.

Esta máscara se compone de cuatro bytes sin tener en cuenta el último. Cada uno de estos (3*8=) 24 bits define una línea del modo GRAPHICS 0. El bit es activado al comenzar una línea lógica en una determinada línea física.

Líneas físicas relacionadas con los bits de los bytes 690 a 692.

byte	bit	7	6	5	4	3	2	1	0
690		0	1	2	3	4	5	6	7
691		8	9	10	11	12	13	14	15
692		16	17	18	19	20	21	22	23

Identificación (flag) de los caracteres invertidos, inicializada a 0. Al pulsar la tecla INVERS (: tecla "ATARI") se activa el bit 7 (128).

El motor del display relaciona los valores de los caracteres entrados a través del teclado con el valor de este registro mediante la operación lógica OR. Mientras INVFLG sólo tome el valor 0 o 128, el valor propio del carácter no se modifica, puesto que el juego de carácter sólo consta de 128 elementos. Al sumar 128 al valor del carácter se representa el carácter inverso.

Sin embargo, también pueden colocarse otros valores (= bit tipo) en este registro e invertir así los bits 6 a 0 de los caracteres llamados. De esta forma se modifica el valor decimal y aparecerá el carácter de este valor en lugar del

correspondiente a la tecla pulsada. ¡Sin embargo, hay que tener en cuenta que Invertirlo se refiere al código del teclado y no al ATASCII!

Resumiendo: se combina mediante OR el bit tipo del código de teclado correspondiente al carácter llamado con el bit tipo de INVFLG. A continuación se convierte el valor resultante en el correspondiente carácter ATASCII a emitir en la pantalla. INVFLG sólo sirve para entradas en modo directo. Se debe escribir en este registro antes de efectuar la entrada.

El siguiente programa permite convertir su ATARI en una máquina de escribir con letra secreta:

```
0 REM ADDR694.BEC
1 REM *****
2 REM * *
3 REM * ESCRITOR DE LETRA OCULTA *
4 REM * *
5 REM *****
10 DIM T$(1):PRINT CHR$(125)
20 TRAP 20
30 PRINT :PRINT "ENTRE POR FAVOR MODALIDAD DE GRAFICOS"
40 PRINT :PRINT "(DESDE 1 HASTA 27)":PRINT
50 INPUT J
60 IF J<1 THEN 20
70 IF J>127 THEN 20
80 POKE 694,J
90 TRAP 90
100 PRINT :PRINT "ESCRIBA POR FAVOR:":PRINT :PRINT
110 INPUT T$
```

10: El DIMensionado de T\$ es aleatorio, puesto que no causará efecto alguno durante el programa.

30 a 70: J toma el valor decimal que será POKEado hacia INVFLG. El valor 128 no es interesante, ya que sólo sirve para llamar a la representación inversa del mismo carácter. Valores mayores que 128 causan el mismo efecto que el valor resultante de restarles 128; los bits tipo son idénticos, pero el bit 7 está activado en un caso y desactivado en otro. Sin embargo, el activado/desactivado del bit 7 puede efectuarse en cualquier momento pulsando la tecla INVERS.

80: Se POKEa J.

110: Ahora puede martirizar las teclas a su gusto y se asombrará de los caracteres que aparecerán en la pantalla. ¡Pruebe también con SHIFT, CONTROL e INVERS! El programa debe finalizarse con RETURN.

695 \$2B7 FILFLG

Identificación FILL del comando DRAWTO. Si el proceso que se está ejecutando es un DRAWTO, el registro tiene valor 0; si es un FILL (comando XIO 18), el valor es diferente de 0.

696-698 \$2B8-\$2BA TMPROW/COL

Registro intermedio utilizado por ROWCRS y COLCRS.

699 \$2BB SCRFLG

Identificación (flag) para el scroll temporal. Cuenta la cantidad de líneas físicas menos 1 que han sido borradas en el borde superior de la pantalla. Puesto que una línea lógica está formada por hasta tres líneas físicas, SCRFLG puede tomar valores de 0 a 2.

El efecto scroll de la ventana de texto es como un desplazamiento hacia arriba de toda la pantalla de GRAPHICS 0 en la memoria (;800 bytes!). Ello puede provocar superposiciones de datos, como por ejemplo datos gráficos de PM, juegos de caracteres, etc. (), depositados anteriormente por encima de RAMTOP. En caso de duda es aconsejable reservar una zona libre de 800 bits por encima de RAMTOP, permitiendo así el desarrollo del scroll.

700, 701 \$2BC, \$2BD -

Registro intermedio durante el proceso FILL.

Identificación (flag) de las teclas SHIFT y CONTROL.

Bit 7 activado: se ha pulsado la tecla STRL y se emiten tanto los códigos de control como los pseudo-caracteres gráficos. Para representar letras (mayúsculas y generales) el bit 7 debe estar desactivado.

Bit 6 activado: tecla SHIFT pulsada. En estado normal, la tecla SHIFT siempre está pulsada, puesto que SHFLOK se inicializa con 64. De aquí viene el nombre del registro: SHIFT-Lock. Con la tecla CAPS (: CAPS LOWR) el bit 6 se puede poner a 0, trabajando así como una máquina de escribir. En este modo habrá que pulsar pues la tecla SHIFT para escribir en mayúscula.

Indica la cantidad de posibles líneas de texto para PRINT.

24 es el valor normal de GRAPHICS 0

4 es el valor correspondiente a la ventana de texto

0 se encuentra sin ventana de texto en todos los modos gráficos. Aquí se incluyen los modos de caracteres en color (GRAPHICS 1, 2, 12, 13).

A través de este registro el manejador del display comprueba si se ha reclamado pantalla partida (gráfica/texto). También sirve para crear una ventana de texto (POKE 703,4) en GRAPHICS 0, pudiendo escribirse en las veinte líneas superiores únicamente mediante la instrucción PRINT dirigida a la ventana gráfica (PRINT 6). Esta parte superior no se altera al realizarse el scroll de la ventana de texto.

Esta técnica se utiliza en el menú del DOS.

Registro de color del jugador (player) 0 y proyectil (missile) 0.

Los cuatro registros de color PCOLR0-3 se leen únicamente en gráfico PM y en el modo GITA GRAPHICS 10. El comando BASIC SETCOLOR no alcanza los registros 704 a 707, por, lo que los valores de color sólo pueden ser POKEados hacia ellos.

El ATARI puede crear dieciséis matizes diferentes de colores, identificados por los números 0 a 15. Sin embargo, a diferencia de otros homecomputers de parecidas condiciones en cuanto a precio, el ATARI también dispone de dieciséis gradaciones de claridad diferentes desde 0 = oscuro a 15 = claro, resultando así 256 valores de color diferentes. Sin embargo, sólo se pueden representar de 1 a un máximo de 16 colores distintos en una pantalla, según el modo gráfico utilizado (sin hacer uso de DLI).

El valor del color se determina mediante la fórmula:

Valor de color = matiz * 16 + gradación de claridad

Esto significa que el matiz de color (de 0 = 0000 a 15 = 1111) ocupa el nibble superior, mientras que la gradación de claridad (también de 0 = 0000 a 15 = 1111) ocupa el nibble inferior de los registros de color.

Puesto que no se lee el bit 0 de los registros de color, se dispone en realidad de sólo ocho gradaciones de claridad (valores pares). Únicamente en el modo GITA GRAPHIC 9 se muestran las dieciséis gradaciones posibles.

Al activar el bit 5 (tercer color por superposición) del registro GPRIOR (623 = \$26F), el valor del color correspondiente a este matiz de superposición resulta de una combinación mediante el OR lógico.

primer valor de color	rosa = decimal	52	= binario
0011 0100			
segundo valor de color	verde = decimal	226	= binario
1110 0010			
combinado con OR =	marrón = decimal	246	= binario
1111 0110			

A consecuencia de la combinación mediante la operación lógica OR, el matiz de la zona superpuesta siempre tiene una gradación de color superior a los colores aislados, es decir un matiz de color con mayor número de identificación y mayor claridad.

705-707 \$2C1-\$2C3 PCOLR1-3

Valor de color para el jugador (player) 1 a 3 y proyectil (missile) 1 a 3.

708 \$2C4 COLOR0

Los registros de color siguientes se leen en los diversos modos gráficos. Pueden ser escritos con el comando BASIC SETCOLOR que tiene el formato:

SETCOLOR r,f,h

r puede tomar valores de 0 a 4 y hace referencia a los registros de color COLOR0 (708 = \$2C4) a COLOR4 (712 = \$2C8).

f puede tomar valores de 0 a 15 y corresponde a los dieciseis matices de color estándar.

h puede tomar valores pares entre 0 y 14 y determina la claridad.

El comando SETCOLOR es seguramente el más inútil de todo el BASIC del ATARI, puesto que es más difícil de utilizar que POKE, con el que se consigue el mismo resultado. Puesto que los valores de r y f para SETCOLOR se deben hallar de forma experimental, también puede escribirse directamente el valor de color (f*16+h) en el registro. Al tener que definir varios colores, con el comando POKE se ahorra bastante memoria en los programas BASIC.

709-712 \$2C5-\$2C8 COLOR1-4

Registros de color 1 a 4.

En los diferentes modos gráficos, los registros de color tienen distintas misiones y se accede a ellos mediante varios comandos BASIC COLOR. En el apéndice memoria de pantalla podrá encontrar un cuadro resumen al respecto.

Al conectar el ordenador, los registros COLOR toman valores diferentes, elegidos, según criterios técnicos y no estéticos (de difícil apreciación), para conseguir un contraste óptimo.

Registro	valor de color	matiz de color	claridad
708 COLOR0	40	2	8
709 COLOR1	202	12	10
710 COLOR2	148	9	4
711 COLOR3	70	4	6
712 COLOR4	0	0	0

El siguiente programa es un modesto ejemplo de los efectos que se pueden conseguir mediante el acceso a los registros de color:

```

0 REM COLPOK.BEC
1 REM *****
2 REM * *
3 REM * CAMBIO RAPIDO DE COLOR *
4 REM * *
5 REM *****
10 GRAPHICS 10
20 FOR J=1 TO 8
30 POKE 704+J, J*4
40 NEXT J
50 FOR C=0 TO 8
60 COLOR C
70 FOR I=0 TO 7
80 PLOT C*8+I,0:DRAWTO C*8+I,191
90 NEXT I
100 NEXT C
200 A=PEEK(705)
210 FOR J=0 TO 6
220 POKE 705+J,PEEK(706+J)
230 REM FOR W=0 TO 200:NEXT 2
240 NEXT J
250 POKE 712,A
260 GOTO 200

```

Con motivo de una mayor elección se ha utilizado el modo gráfico 10, que además del color de fondo admite otros ocho matices de libre definición.

En este programa los valores de color se encuentran en los registros 705 a 712. El 704 define el color de fondo y se activa a 0 (= negro).

20 a 40: El bucle FOR-NEXT escribe valores de color desde 4 (1*4) hasta 32 (8*4) en los registros de color 705 (704+1) a 712 (704+8). No fue necesario escribir en el registro 704, puesto que su valor default 0 corresponde al color deseado. Naturalmente también se puede asignar un valor cualquiera a cada registro de color por separado. Sin embargo, para conseguir el efecto deseado son preferibles los matices graduados por claridad.

50 a 100: Llenan la pantalla de GRAPHICS 0 con barras verticales coloreadas y ordenadas según los registros de color.

200: En A se guarda el valor de color del registro 705.

210: A continuación se cambian los valores de color de siete registros (J=0 TO 6),

220: escribiendo en el registro precedente el valor del siguiente.

250: Finalmente el último registro (712) adquiere el valor del primero (705), contenido en A.

Si ejecuta el programa de esta manera se produce el efecto de un tambor giratorio. La velocidad de giro se reduce eliminando el REM de la línea 230. Al aumentar el valor del

contador W del bucle, el "giro" del tambor se hace más lento.

Otro efecto eficaz consiste en cambiar el valor de color de un registro con una secuencia corta:

```
260 X=X-2:IF X<0 THEN X=254
270 POKE 704,X
280 GOTO 200
```

Añadiendo estas tres líneas al listado anterior, el fondo (704) empieza a brillar en varios colores. Si desea conocer la velocidad con que cambian los valores de color utilizando POKE, modifique el destino del salto en la línea 280 de 200 a 260.

729 \$2D9 KEYDEL

Determina el tiempo que tarda en activarse la función repetidora del teclado. El valor default es 40. Con el valor 0 se interrumpe la función repetidora. Al incrementar los valores en orden creciente desde 1 a 255, aumenta su retardo. Cuando tiene valor 1 el retardo es tan corto, que al pulsar una tecla resulta imposible escribir un único carácter en la pantalla. (sólo XL)

730 \$2DA KEZREP

Cuando se activa la función repetidora después de pasar por KEYDEL, KEYREP determina la frecuencia de repetición. El valor default es 5. La repetición es tanto más rápida cuanto menor sea el valor del registro. Si el valor asignado es 0, sólo puede darse una única repetición. (sólo XL)

731 \$2DB NOCLIK

Flag del clic del teclado. El valor 0 evita el clic, mientras que cualquier otro valor lo activa. (sólo XL)

732 \$2DC HLPFLG

Flag de la tecla HELP. Su valor default es 0. Al pulsar la tecla HELP se activan los bits 0 y 4 (= 17). Si se pulsa simultáneamente con SHIFT, se activa además el bit 6, con lo cual el registro contendrá el valor 81. HELP y CONTROL activan adicionalmente el bit 7 (= 145). Al pulsar SHIFT y CONTROL al mismo tiempo, el accionamiento de HELP no tendrá consecuencias. ¿Los bits activados no se vuelven a poner a 0 al anular las teclas! (sólo XL)

740 \$2E4 RAMSIZ

Tamaño de la RAM disponible. Contiene tan sólo el byte HI, indicando pues el tamaño de la RAM en páginas. Contiene el mismo valor que RAMTOP (106 = \$6A).

741,742 \$2E5,\$2E6 MEMTOP

Puntero hacia el límite superior de la memoria RAM disponible. Este valor es renovado con cada powerup o RESET, así como por cada comando GRAPHICS y OPEN, que se dirige hacia el display.

743,744 \$2E7,\$2E8 MEMLO

Puntero hacia el límite inferior de la memoria RAM libre. Es modificado por ejemplo por el DOS, cargado en la zona

inferior de la memoria.

MEMLO apunta hacia la primera posición libre de memoria que puede ser ocupada por un programa de aplicaciones.

750,751 \$2EE,\$2EF CBAUD

Cóntiene la velocidad de transferencia de datos en baudios para el cassette. Se inicializa a 1484, que corresponde a una velocidad de transferencia de 600 (bits/segundo), y es ajustada por el SIO. Cada grabación en cassette comienza con un tipo de bits de encendido/apagado en el que el motor del cassette corrige la velocidad de transferencia de datos en baudios.

752 \$2F0 CRSINH

Flag de la supresión del cursor. Se activa a 0 en powerup, RESET, BREAK y OPEN "S:" o "E:", que visualiza el cursor. Cualquier valor diferente a 0 da lugar a que el cursor desaparezca después del siguiente movimiento.

753 \$2F1 KEYDEL

Flag del retardo de teclas. Tiene el valor 0 cuando no hay ninguna tecla pulsada. En caso contrario toma el valor 3, que disminuye en 1 con cada VBLANK. La próxima entrada del teclado no será admitida hasta alcanzar nuevamente el valor 0. KEYDEL determina, pues, el intervalo de tiempo entre dos pulsaciones.

736-737 INIT DOS FILE

738-739 RUN DOS FILE

754

\$2F2

CH1

Contiene el valor de la tecla pulsada anteriormente, cargado de CH (764 = \$2FC) hacia aquí.

755

\$2F3

CHACT

Registro utilizado para representar caracteres.

Si tiene valor 0, los caracteres inversos se representan en forma de normales. Con ello se consigue que el cursor sea invisible.

El valor 1 da lugar a que todos los caracteres inversos se emitan como espacios. El cursor también será invisible, no obstante desaparecerá el carácter que se encuentra bajo él.

El valor default es 2.

Utilizando el valor 3, todos los caracteres inversos se representan en forma de espacio inverso.

Al activar el bit 2 (= 4) todos los caracteres se emiten invertidos.

En el modo de caracteres GRAPHICS 1, 2, 14 y 15 los respectivos valores simplemente dan lugar a que los caracteres se emitan invertidos.

Pero vea usted mismo el efecto que los diversos valores de este registro causan en GRAPHICS 0:

```

0 REM ADR755.BEC
1 REM *í*****í*****
2 REM *
3 REM * CARACTERES INVERTIDOS *
4 REM *
5 REM *****
10 GRAPHICS 0:POKE 703,4
20 ? "321 321 CBA CBA"
30 FOR J=0 TO 255
40 POKE 755,J
50 ?# 6;J;" ";
60 OPEN #1,4,0,"K:"
70 GET #1,D
80 CLOSE# 1
90 NEXT J

```

```

*****
* ¡Atención! ¡ El listado contiene caracteres inversos *
* absolutamente imprescindibles para su comprensión! *
*****

```

10: En GRAPHICS 0 se POKEa una ventana gráfica.

20: Se muestran con PRINT caracteres normales e inversos en la ventana gráfica.

40: El valor es POKEado hacia 755.

50: Como medida de control, en la ventana de texto aparece el valor recién POKEado.

60 a 80: Cuando el usuario haya pulsado cualquier tecla, el programa continúa colocando el siguiente valor en 755.

Como puede ver, en este registro sólo actúan los bits 0 y 2. Gracias a las ocho regulaciones diferentes se pueden crear gran número de bonitos intermitentes, conmutando entre dos valores de CHACT. El siguiente programa le muestra las 64 combinaciones posibles:

```

0 REM ADR755B.BEC
1 REM *****
2 REM *                                     *
3 REM *   INTERMITENTE 8X8                 *
4 REM *                                     *
5 REM *****
10 GRAPHICS 0:POKE 703,4
20 POSITION 5,12
30 PRINT# 6;"INTERMITENTE! INTERMITENTE! INTERMITENTE!
INTERMITENTE!"
40 FOR I=0 TO 7
50 FOR J=0 TO 7
60 PRINT I;"/",J:PRINT
70 POKE 755,I
80 FOR W=0 TO 100:NEXT W
90 POKE 755,J
100 FOR W=0 TO 100:NEXT W
110 IF PEEK(764)=225 THEN 70
120 OPEN#1,4,0,"K:"
130 GET# 1,D
140 CLOSE# 1
150 NEXT J
160 NEXT I

```

```

*****
* ;Atención! ;El listado contiene caracteres inversos *
* absolutamente imprescindible para su comprensión! *
*****

```

10 a 30: Se imprime el texto en el centro del gráfico 0.

40 y 50: Los bucles indentados procesan todas las ocho por ocho combinaciones entre las que se puede conmutar en CHACT.

60: Muestra la combinación actual en la ventana de texto como medida de control.

80 y 100: Aquí se puede modificar la frecuencia de emisión intermitente.

110 a 130: La siguiente combinación empieza a centellar al pulsar cualquier tecla del keyboard.

Puntero hacia la dirección inicial del juego de caracteres estándar del ATARI.

CHBAS es simplemente un puntero al byte HI; para obtener la verdadera dirección hay que multiplicar 256 por el valor aquí indicado. El valor default de este registro es 224. El juego de caracteres está formado por 128 elementos. Los caracteres (ATASCII 128 a 255) Inversos (negativos) no precisan de datos especiales, creándose simplemente al invertir el bit tipo del carácter normal. El flag de la representación inversa es el bit 7 (= 128). Esto significa, que el valor ATASCII de un carácter Inverso resulta de sumarle 128 a su correspondiente carácter normal.

En los modos gráficos GRAPHICS 1 y 2 se dispone tan sólo de la mitad del juego de caracteres, es decir, de signos de puntuación, cifras y letras mayúsculas. Al cambiar CHBAS al principio de la parte posterior del juego de caracteres, también pueden emitirse letras minúsculas. Sin embargo, es imposible representar al mismo tiempo mayúsculas y caracteres comunes en una pantalla (sin modificar el propio juego de caracteres). Se puede cambiar el puntero utilizando POKE 756,226.

El juego de caracteres estándar corresponde naturalmente al americano. Sin embargo, y gracias a la colonización electrónica, los XL también cuentan con un juego de caracteres europeo, accesible con POKE 756,204.

En el apartado "juego de caracteres" del apéndice se explica la configuración del mismo, así como la forma de diseñarlo, grabarlo y cargarlo.

760 \$2F8 ROWINC

Signo (+1 o bien -1) de DELTAR (118 = \$\$76). (≠: 121 = \$79)

761 \$2F9 COLINC

Signo de DELTAC (119,120 = \$77,\$78). (≠: 122 = \$80)

762 \$2FA CHAR

Contiene el código interno del último carácter leído o escrito. Por lo general, el BASIC es demasiado lento para leer este registro, por lo que el resultado de PEEK (762) suele ser sólo el valor 128 (cursor visible = espacio Inverso) o 0 (cursor invisible = espacio).

763 \$2FB ATACHR

Contiene el valor ATASCII del último carácter escrito o leído y es utilizado al convertir el ATASCII en código Interno. Al trabajar en modo gráfico, se coloca en este registro el valor del color del punto gráfico en cuestión, y en los comandos X10 17 (DRAW) y X10 18 (FILL) el color de la línea a dibujar.

764 \$2FC CH

Contiene el código de teclado correspondiente a la última tecla pulsada. Permite leer entradas de teclado utilizando PEEK (764) sin necesidad de entrar RETURN.

El registro contiene un 255 mientras no haya ninguna tecla pulsada. En la tabla siguiente se muestran los códigos de teclado de las teclas restantes. Hay que sumar los valores decimales de los lados superior e izquierdo para obtener el código del carácter que se encuentra en su intersección:

	0	1	2	3	4	5	6	7
00	L	J	;			K	+	*
08	O		P	U	return		-	=
16	V		C			B	X	Z
24	4		3	6	escape	5	2	1
32	,	espacio	.	N		M	/	inverso
40	R		E	Y	Tab	T	W	Q
48	9		0	7	backs	8	()

Estos caracteres ocupan los bits 0 a 5. Independientemente de ello, la pulsación simultánea de cualquier tecla con SHIFT activa el bit 6 y el bit 7 con CONTROL. Pulsar al mismo tiempo las teclas SHIFT y CONTROL no está permitido y es ignorado por el OS.

El código de teclado de un carácter corresponde al offset del mismo en la tabla de definiciones del teclado, utilizada para convertir el código de teclado en ATASCII. El usuario puede definir una tabla de definiciones propia y asignar nuevos valores ATASCII aleatorios a cada tecla. Para ello es necesario que la tabla conste de tres partes: al primer tercio de 64 bytes se accede a través de las letras minúsculas; los 64 bytes siguientes a través de la misma tecla + SHIFT; utilizando la tecla + CONTROL se accede a los últimos 64 bytes.

Después de colocar la nueva tabla de definiciones en la memoria, hay que cambiar el puntero KEYDEF (12,122 = \$79,\$7A) para que apunte hacia el principio de la nueva tabla.

Las teclas RESET, BREAK, SHIFT, CONTROL, así como las teclas de funciones OPTION, SELECT, START, HELP y la combinación CONTROL + "1" no se procesan a través de esta tabla y, por lo tanto, no se puede cambiar su valor en este registro.

Una aplicación de CH se muestra en la línea 110 del programa ejemplo ADR755B.BEC.

765 \$2FD FILDAT

Valor COLOR correspondiente a la zona a llenar por X10 18 (FILL).

El siguiente programa le muestra la forma de emplear el comando FILL:

```
0 REM ADR765.BEC
1 REM *****
2 REM *
3 REM * X10 18=COMANDO-FILL
4 REM *
5 REM *****
10 GRAPHICS 31
20 POKE 708,4:POKE 709,50:POKE 710,1
30 POKE 765,1
40 COLOR 2
45 REM PLOT 80,80
50 PLOT 139,95
60 DRAWTO 109,0
70 DRAWTO 49,0
80 POSITION 19,95
90 X10 18,#6,0,0,"S:"
95 GOTO 95
100 PLOT 109,191
110 DRAWTO 139,96:REM COLOR 1
120 DRAWTO 19,96
130 POSITION 49,191
140 X10 18,#6,0,0,"S:"
150 GOTO 150
```



```

160 COLOR 3
170 PLOT 159,191
180 DRAWTO 159,0
190 DRAWTO 0,0
200 POSITION 0,191
210 POKE 765,2
'220 X10 18,#6,0,0,"S:"
230 GOTO 230
240 PLOT 159,191:REM COLOR 2
250 DRAWTO 159,0
260 DRAWTO 110,0
270 COLOR 2:POSITION 140,95
280 X10 18,#6,0,0,"S:"
290 PLOT 159,96
300 DRAWTO 140,96
310 POSITION 110,191
320 X10 18,#6,0,0,"S:S"
330 GOTO 330

```

10: Se conecta GRAPHICS 15 sin ventana de texto (+16). Si posee un ATARI antiguo incapaz, de dirigirse a este modo a través de GRAPHICS, utilice GRAPHICS 7+16. En este caso será necesario adaptar parcialmente los PLOTs.

20: Valores de color para COLOR 1, 2 y 3. COLOR 0, que corresponde al fondo y está contenido en el registro 712, mantiene su valor default 0 (= negro).

30: En FILDAT se determina el valor de COLOR 1 para X10 18.

40: Se llama el color 2 destinado a los PLOTs siguientes.

50 a 80: La función FILL precisa de las señalizaciones siguientes: una línea PLOTeadas como límite derecho; una línea PLOTeadas como borde superior; ambas definen tres esquinas de la superficie a llenar. con FILL: inferior derecha, superior derecha y superior izquierda; como última

orientación se precisa el punto inferior izquierdo, que es entrado en forma de comando POSITION. Con el comando FILL sólo se pueden rellenar superficies cuadradas, aunque de forma cualquiera. La única condición adicional que debe cumplirse es que la esquina superior izquierda se encuentre por encima de la superior derecha, puesto que

90: el comando FILL realiza ahora lo siguiente: desde la esquina superior izquierda se dibuja (PLOT) una línea del color requerido actual hacia el punto indicado por POSITION. Después de cada uno de los puntos así PLOTeados, se traza una línea con el color colocado en FILDAT a partir de este punto hacia la derecha, hasta que un punto ya PLOTeadado llegue al límite derecho. A continuación se dibuja el siguiente punto del lado izquierdo del cuadrado, se vuelve a trazar la línea FILL hacia la derecha, etc.

Para demostrar este efecto, añada al listado la línea siguiente:

```
45 PLOT 80,80
```

y escriba RUN. Verá que cuando la rutina FILL llega a la línea 80, ésta se llena sólo hasta la columna 79, mientras que el resto de la línea se mantiene del color habitual.

95: Vuelva a borrar ahora esta línea y la 45 para continuar.

100 a 140: El hexágono a dibujar en la pantalla debe partirse en dos trapecios. A continuación se muestra la parte inferior.

110: Si ejecuta ahora el programa, podrá ver claramente como se va trazando (PLOT) primero el lado izquierdo y superior del cuadrado. Durante el FILL se crea además el borde izquierdo. El cuadrado se limita hacia abajo por el borde de la superficie de FILL. Si desea obtener un marco coloreado alrededor, habrá que dibujar (PLOT) además el borde inferior del color deseado. En caso contrario, verá

que el borde superior de la mitad inferior del hexágono se emite del COLOR válido para PLOT, trazándose así una línea divisoria horizontal en el centro del hexágono. Si prefiere eliminar esta línea, borre el REM de la línea 110, activando el comando COLOR. Se permite determinar el mismo valor de COLOR para PLOT y FILL.

150: Después de apreciar el efecto de estas manipulaciones, elimine esta línea para acceder a la parte posterior del programa.

160 a 220: Se encargan de dibujar (PLOT) de COLOR 3 los bordes exteriores de la ventana gráfica como líneas que limiten el comando FILL. Sin embargo, cuando el comando X10 18 se ejecute utilizando COLOR 2, considerará los bordes izquierdos ya existentes del rectángulo como límite derecho. Por lo tanto, no se rellenará la pantalla restante sino únicamente la superficie entre el lado izquierdo y el hexágono.

230: Cuando haya estudiado este proceso, borre esta línea de captura.

240 a 280: Para rellenar ahora la superficie restante que se encuentra a la derecha del hexágono habrá que realizar más esfuerzos, ya que es imposible colorear esta superficie con ayuda de un comando. Si se dibujasen (PLOT) los límites izquierdo y superior de la superficie (margen de pantalla) y se situase después el cursor con POSITION en el extremo inferior izquierdo, se trazaría tan sólo el borde izquierdo. Este borde se encuentra en el interior del hexágono. El comando FILL comenzaría por el interior del hexágono y consideraría sus bordes derechos como límite derecho. Sin embargo, una vez se haya rellenado una superficie con FILL, ésta no podrá volver a modificarse utilizando FILL.

La superficie restante de la derecha deberá rellenarse pues en dos partes. Las líneas 240 a 280 se encargan de la mitad superior. El molesto borde izquierdo puede adaptarse en cuanto a color eliminando el REM de la línea 240.

El trazado del límite derecho (margen de pantalla) es innecesario, puesto que ya se ha efectuado en las líneas 100 y 110 al intentar sin éxito rellenar de una vez la totalidad de la superficie restante con FILL.

766

\$2FE

DSPFLG

Flag del display para tratar los caracteres de control. Los caracteres de control tales como movimiento del cursor, borrado de pantalla, INSERT, DELETE, BACKS o TAB pueden introducirse en programas BASIC de dos formas diferentes. La primera:

```
PRINT CHR$(n)
```

ejecuta la función mencionada al sustituir el valor correspondiente por n. Es decir, con n=125 se limpia (CLEAR) la pantalla. Esta instrucción aparece en la mayor parte de los programas ejemplo presentados en este libro. Esta forma tiene la ventaja de que cualquier impresora conectada puede escribirla sin ningún problema.

La segunda posibilidad consiste en programar la función deseada a través de la tecla correspondiente. La instrucción BASIC tendrá la forma mostrada a continuación, donde las indicaciones en () significan que al entrarlo tendrá que pulsar la tecla correspondiente:

```
PRINT "(ESCAPE)(CLEAR)"
```

Al entrar esta instrucción aparece un símbolo gráfico en la pantalla, en este caso se trata de una pequeña flecha inclinada que señala hacia la parte superior izquierda. Para el programador principiante ésta es una instrucción fácil de aplicar, puesto que no necesita acordarse de los valores ATASCII correspondientes a las funciones deseadas sino que, al programar, simplemente deberá pulsar las mismas teclas utilizadas al trabajar en modo directo. Por cierto, la tecla ESCAPE sólo tiene efecto sobre la tecla inmediatamente posterior; si desea entrar varios caracteres de control seguidos, deberá pulsar ESC delante de cada uno de ellos.

Sin embargo, cuando pretenda listar por la impresora su programa elaborado, maldecirá este cómodo método, puesto que la impresora sólo considera los valores recibidos interpretados a su manera, sin estar definidos por la norma ASCII. Es, pues, completamente normal que se vuelva loco. A pesar de ello, un (ESCAPE)(CLEAR), al ir entre paréntesis, se escribe en la impresora. Pero los códigos de valor superior a 128 (por ejemplo, TAB) pueden provocar una reacción sorprendente, diferente en cada modelo de impresora.

Lo mismo sirve para los pseudo-caracteres gráficos cuyos valores ATASCII oscilan entre 0 y 31, que son los caracteres de control más importantes para la impresora (norma ASCII); la impresora empieza luego a realizar saltos de TAB, avances de línea y formulario, retorno del carro (carriage return) y de vez en cuando suena la campanita. ATARI no ha conseguido todavía ofrecer una impresora capaz de imprimir todo el juego de caracteres ATASCII. Actualmente no existe siquiera una interfase adecuada a tal fin. Se aconseja, pues, sustituir todos estos caracteres por sus correspondientes comandos CHR\$ -cuyo efecto es idéntico al de los caracteres pseudo-gráficos o de control entrados en forma de strings- en aquellos programas que deberán imprimirse.

Si el valor de DSPFLG es diferente de 0, el carácter de control en cuestión se representa en la pantalla por su gráfico correspondiente, o sea, al igual que si se hubiese pulsado ESC. 0 es el estado normal de este registro.

767

\$2FF

SSFLAG

Flag de inicio/parada de la salida a pantalla. Mientras su valor sea 0 la salida se realiza sin problemas. El valor 255 (inverso de 00000000), interrumpe la salida a pantalla y el sistema espera hasta que se coloque un 0 en este registro. SSFLAG se conmuta pulsando simultáneamente CONTROL y "1".

768 \$300 DDEVIC

Código de identificación para el flag de los periféricos.

769 \$301 DUNIT

Número de periférico; puede ser definido por el usuario.

770 \$302 DCOMND

Byte de comando. Contiene el número del flag que corresponde a la operación a realizar.

771 \$303 DSTATS

Estado de dispositivo.

772,773 \$\$304,\$305 DBUFLO/HI

Dirección del buffer de datos.

774 \$306 DTIMLO

Tiempo de espera para el motor del dispositivo hasta producirse el mensaje de error timeout.

775 \$307 DUNUSE

Byte libre.

776,777 \$308,\$309 DBYTLO/HI

Cantidad de bytes que se encuentran en el buffer al que apunta DBUFLO/HI.

778,779 \$30A,\$30B DAUX1/2

Informaciones auxiliares. En operaciones de diskette, por ejemplo, para indicar LO y HI del número de sector.

780,781 \$30C,\$30D TIMER1

Timer de la velocidad de transferencia de datos en baudios.

783 \$30F CASFLG

Es diferente de 0 cuando la operación que se está realizando se dirige al cassette.

784,785 \$310,\$311 TIMER2

Timer 2 de intervalo. TIMER1 y TIMER2 se utilizan para ajustar la velocidad de transferencia de datos, en baudios, de las operaciones con cassette. Se encargan de comparar el valor estándar con el bit tipo que se encuentra al principio de un fichero de cassette. La diferencia de los timer sirve para hallar en una tabla el valor de corrección, a través del cual se transmitirá entonces la velocidad de transferencia en baudios al CBAUDL/H(750,751) = \$2EE,\$2EF).

791 \$317 TIMFLG

Flag del timeout.

792 \$318 STACKP

Puntero de stack S10.

793 \$319 TSTAT

Registro intermedio reservado para informaciones sobre el estado del S10.

794-831 \$31A-\$33F HATABS

Tabla de las direcciones del motor. Sirve para asignar los periféricos a sus respectivos motores. Se pueden registrar hasta 38 entradas de tres bytes cada una. El primer byte contiene el nombre del dispositivo (C, D, E, K, P, S, R) en código ATASCII; los bytes dos y tres contienen LO y HI de la dirección inicial del motor. Los bytes libres tienen valor 0.

832-847 \$340-\$34F IOCBO

Bloque de control 0 del I/O. Es utilizado normalmente para el editor de pantalla ("E:"). En los modos GRAPHICS el canal 0 está abierto hacia la ventana de texto. Si no existe ninguna ventana de texto y se abre el canal 0, toda la pantalla se convierte en modo editar (GRAPHICS 0). Esto ocurre por ejemplo para mostrar READY cuando un programa gráfico ha sido procesado sin ventana de texto.

Los comandos BASIC NEW y RUN cierran todos los canales excepto el 0. Al abrir un canal hacia "S:" o "E:", se borra toda la pantalla.

Si en un programa la entrada efectuada por el usuario debe decidir si los datos tienen que salir en pantalla o en impresora, se puede evitar la programación reiterada de todas las posibles salidas en forma de instrucciones PRINT y LPRINT. Basta con modificar el vector de las direcciones 838 (= \$340) L0 y 839 (= \$34F) H1. Habitualmente, este puntero contiene los valores 163 y 246. Si aquí se colocan 166 L0 y 238 H1, se emitirá hacia la impresora lo que normalmente va hacia el editor.

842 \$34A ICAX1

Esta dirección tiene un significado especial porque posibilita leer en la pantalla. Su valor habitual es 12, estado en el que se escribe en pantalla. Si el valor depositado es 13, se conecta el modo de la tecla RETURN que permite la lectura en la pantalla.

Esto funciona de la siguiente forma:

```

0 REM ADR842.BEC
1 REM *****
2 REM *
3 REM * LECTURA DESDE LA PANTALLA *
4 REM *
5 REM *****
10 GRAPHICS 0:POSITION 2,4
20 PRINT 1000
30 FOR W=0 TO 500:NEXT W
40 PRINT 2000
50 FOR W=0 TO 500:NEXT W
60 PRINT 3000
70 FOR W=0 TO 500:NEXT W
80 PRINT "4000REM**ESTA LINEA SE INCORPORA EN EL PROGRAMA
COMO LINEA DE BASIC LEYENDO DESDE LA PANTALLA**"
90 FOR W=0 TO 500:NEXT W
100 PRINT "CONT"
110 FOR W=0 TO 500:NEXT W
120 POSITION 2,0
130 FOR W=0 TO 500:NEXT W
140 POKE 842,13:STOP
150 FOR W=0 TO 500:NEXT W
160 POKE 842,12
170 FOR W=0 TO 500:NEXT W
4000 REM **ESTA LINEA SE INCORPORA EN EL PROGRAMA COMO LINEA
DE BASIC LEYENDO DESDE LA PANTALLA**

```

10: El comando GRAPHICS da lugar al borrado de pantalla. Al fin y al cabo sólo se pretende leer en la pantalla lo que el programa escribe en ella. Es importante comenzar a escribir en la pantalla lo suficientemente abajo como para evitar que el mensaje del sistema "STOPPED ..." oculte lo escrito.

20: Se muestra el número 1000 en la pantalla. Más tarde, cuando lea en ella, el BASIC lo considera como número de

línea y, puesto que no le sigue ninguna instrucción, se borrará la línea 1000 de la memoria.

30: El bucle de espera sólo sirve para seguir ópticamente la ejecución del programa. Sin embargo, esta línea debería borrarse cuando se proceda a la aplicación real del mismo.

40 a 70: Igual que 20 y 30.

80: Sin embargo, también es posible imprimir (PRINT) una línea completa de programa, que será recogida por el programa al leer en la pantalla.

100: Para finalizar hay que escribir "CONT" para conseguir que el programa continúe su ejecución después de la lectura.

120: Una vez escrito en pantalla todo lo que deberá ser leído en ella, hay que emplazar el cursor invisible por encima de ello en la pantalla.

140: Ahora se POKe a lectura y se para el programa.

160: Después de leer CONT en la pantalla, el programa continúa su ejecución y se vuelve a POKear ICAX1 a la salida a pantalla.

Si ahora teclea RUN podrá observar tranquilamente cómo se van escribiendo las líneas 1000, 2000, 3000 y 4000. A continuación, el cursor salta hacia la columna 2 de la línea 0 donde emite su "STOPPED IN LINE 140". Se inicia el proceso de lectura, que finaliza cuando READY anuncia el final del programa en la parte inferior. Entonces teclee LIST!

848-863 \$350-\$35F IOCB1

Bloque de control 1 de I/O.

864-879 \$360-\$36F IOCB2

Bloque de control 2 de I/O.

880-895 \$\$370-\$37F IOCB3

Bloque de control 3 de I/O.

896-911 \$380-\$38F IOCB4

Bloque de control 4 de I/O.

912-927 \$390-\$39F IOCB5

Bloque de control 5 de I/O

928-943 \$3A0-\$3AF IOCB6

Bloque de control 6 de I/O.

El comando GRAPHICS abre el canal 6 hacia la pantalla ("S:"). Por esta razón no se puede utilizar el canal 6 después de salir de GRAPHICS 0; es decir, se debe efectuar antes un CLOSE dirigido al canal número 6. Sin embargo, ello nos impide llamar después los comandos PLOT, DRAWTO o LOCATE. Asimismo, los comandos PRINT dirigidos a la ventana gráfica de los modos gráficos 1, 2, 12, 13 y 0 con ventana de texto tendrán que dirigirse al canal 6.

944-959 \$3B0-\$3BF IOCB7

Bloque de control 7 de I/O.

El comando LPRINT utiliza el canal 7. Si este canal está abierto hacia otro periférico, se emite un mensaje de error al efectuar LPRINT. El comando LIST también pasa a través del canal 7, aunque éste ya se haya abierto para otro fin. LIST cierra el canal después de utilizarlo.

960-999 \$3C0-\$3E7 PRNBUF

Buffer de impresora. En él se almacenan los datos antes de transferirlos a la impresora después de EOL.

1001 \$3E9 STRTFLG

Este flag muestra un valor diferente de 0 al pulsar la tecla START durante el powerup.

1021-1151 \$3FD-\$4FD CASBUF

Buffer del cassette.

1152-1791 \$480-\$6FF -

Esta zona está disponible como RAM de aplicaciones para programas cortos en lenguaje máquina. Abarca 640 bytes. Sin embargo, la rutina de coma flotante (floating-point) utiliza las direcciones 1406 a 1535 (= \$57E-\$5FF).

No obstante sólo la página 6 (1536-1791 = \$600-\$6FF) ofrece un lugar de la memoria realmente seguro en esta zona inferior.

1792-5377

\$700-\$1501

FMS

El FMS (file management system) es una parte del DOS que crea una conexión entre el BASIC o DUP y la unidad de disco.

5440-

\$1540-

DUP

El DUP (disk utilities package) administra diversas funciones del DOS, como por ejemplo "Copy".

La memoria necesaria para ello puede variar, llegando hasta la dirección señalada por MEMLO, como máximo hasta 13062 (= \$3306).

Al llamar utilidades DUP del menú del DOS, pueden producirse escrituras encima de la zona inferior del RAM de aplicaciones. Si el diskette consta de un MEM.SAV, los datos de esta zona se graban en el diskette y se vuelven a colocar en este lugar después de abandonar el DOS. Este proceso es realmente cómodo, pero tiene el inconveniente de que la función MEM.SAV ocupa una parte considerable de los sectores del diskette, y que su ejecución requiere algún tiempo. Por ello se aconseja prescindir del MEM.SAV y realizar las operaciones de diskette utilizando el comando X10, en lugar de realizarlas a través del DOS.

Las siguientes operaciones del DOS pueden sustituirse fácilmente:

Option D; borrar fichero;

X10 33,#1,0,0,"D:nombre de fichero.ext"

Option E; cambiar nombre de fichero;

X10 32,#1,0,0;"D:nombre.viejo, nombre.nuevo"

Option F; proteger fichero;

X10 35,#1,0,0,"D:nombre de fichero.ext"

Option G; desproteger fichero;

X10 36,#1,0,0,"D:nombre de fichero.ext"

Option I; formatear diskette;
XIO 254, #1, 0, 0, "D:"

Option A; la emisión del directorio también puede sustituirse con relativa facilidad. Para ello sólo precisa de dos líneas, que pueden hallarse detrás de cualquier otro programa BASIC:

```
0 REM DIRCPRNT-BEC
1 REM *****
2 REM * *
3 REM *IMPRIMIR DIRECTO.DE DISKETTE*
4 REM * *
5 REM *****
32766 END
32767 CLR :DIM FLN$(18):CLOSE#1::OPEN#1,6,0,"D:*.*":FOR
FLS=0 TO 64:INPUT#1,FLN$:PRINT FLN$:NEXT FLS
```

32766: Bloquea el programa BASIC anterior a esta línea contra el programa directorio.

32767: CLR borra la variable DIMensionada. Ello es necesario para poder acceder reiteradas veces a esta línea.

FLN\$ es DIMensionada en 18 caracteres y recoge el nombre del fichero y la cantidad de sectores ocupados, es decir 18 caracteres.

El comando CLOSE es una medida de precaución en caso de que el canal de datos número 1 acabe de abrirse.

Se abre el canal 1. El número 6 indica como modo operativo del canal la lectura del contenido del diskette. Los "*" se denominan "wild cards" ("comodines"); sustituyen cualquier secuencia de caracteres. " *.*" sustituye cualquier posible nombre de fichero, es decir se leen todos los ficheros

existentes.

El bucle FOR-NEXT repasa todos los ficheros. En un diskette se puede grabar un máximo de 64 ficheros. Aunque después todavía queden sectores libres, no se podrán grabar más ficheros porque ya se ha agotado el directorio. Finalmente se leerá además la cantidad de sectores disponibles.

Sustituyendo FLN\$ por LPRINT FLN\$, la salida puede tener lugar en impresora.

Si el diskette no contiene todavía 58 ficheros, la salida finaliza con el mensaje de ERROR- 136.

GRAFICOS PLAYER - MISSILE

53248-53505 \$D0000-\$D0FF GITA

53248 \$D000 HPOSP0

(W) En este registro se coloca la posición horizontal del jugador (player) 0.

Player (jugador) y missile (proyectil) son objetos gráficos independientes del gráfico normal (playfield = campo de juego) en cuanto se refiere a su forma, color y movimiento. Habitualmente tales elementos se denominan "sprites". Los jugadores del ATARI cubren la altura total de la pantalla en forma de cinta en dirección vertical. La posición horizontal de esta cinta de jugador se determina en HPOSPn.

Las posiciones de los objetos de PM se refieren a la distribución física del tubo de televisión. Por esta razón se admiten valores horizontales de 0 a 277, aunque un jugador entre en la imagen por la izquierda cuando ya haya alcanzado una posición horizontal de 45 y la abandone por la derecha con un valor de 210 aproximadamente. Los valores exactos pueden variar ligeramente de un dispositivo a otro.

La posición vertical de un jugador se define por la situación de sus datos. Cada objeto de PM dispone de una zona de memoria separada. En PMBASE (54279 = \$D407) se explica la forma en que se organiza toda la memoria de PM. Un player siempre ocupa un byte de anchura; sin embargo, según definición hecha en el correspondiente registro SIZE0 (53256 = \$D008) cada bit puede representar dos, cuatro u ocho puntos de imagen, lo que da lugar a diversas anchuras de display del jugador.

Puesto que el jugador ocupa la altura completa del tubo de televisión, abarca 128 bytes, si cada byte ocupa dos líneas de pantalla (resolución de dos líneas), o 256 bytes (resolución de una línea). Si el jugador no debe ocupar la altura total de la pantalla, se ponen a 0 los bytes correspondientes. La manera de convertir un byte en una disposición de puntos de imagen se explica en el apéndice juego de caracteres.

Este y muchos de los registros que siguen a continuación tienen funciones diversas de lectura (R) y escritura (W). Después de colocar aquí la posición horizontal de jugador 0, ésta no podrá ser leída de nuevo con PEEK(53248), puesto que (R) indica si se ha producido una colisión (superposición) entre proyectil 0 y un campo de juego.

El nibble inferior indica en cuál de los campos de juego se ha producido la colisión. Al decir campos de juego nos referimos a los píxel de los diversos valores de COLOR.

Bit	7	6	5	4	3	2	1	0
Valor decimal					8	4	2	1
Campo de juego		no utilizado			3	2	1	0

Después de una colisión entre proyectil y campo de juego (color de juego) 2, PEEK(53248) detecta un 4.

53249-53251 \$D001-\$D003 HPOSP1-3

(W) Posición horizontal de jugador 1 a 3:

(R) Colisión entre - proyectil 1 a 3 / campo de juego.

53252

\$D004

HPOSMO

(W) Posición horizontal de proyectil 0. Los proyectiles se mueven horizontalmente, de igual forma que la expuesta para los jugadores.

(R) Colisiones entre jugador 0 y el campo de juego.

Bit	7	6	5	4	3	2	1	0
Valor decimal					8	4	2	1
Campo de juego	no utilizado				3	2	1	0

Después de una colisión entre jugador 0 y campo de juego (color de juego) 3, PEEK(53252) encuentra un 8.

53253-53255

\$D005-\$D007

HPOS1-3

(W) Posición horizontal de proyectil 1 a 3.

(R) Colisión entre - jugador 1 a 3 / campo de juego.

53256

\$D008

SIZEPO

(W) Determina la anchura de jugador 0.

Un jugador siempre tiene una anchura de un byte (ocho bits). Cada uno de los bits crea un pixel cuando está activado. SIZEPO determina la anchura de un pixel en puntos de imagen. Un valor 0 o 2 de este registro da lugar a la anchura normal de dos puntos de imagen por pixel. Un 1 muestra los pixel con una anchura de cuatro puntos de imagen en la pantalla, mientras que la del jugador será de 32 puntos. Un 3 define la anchura de un pixel en ocho y la de un byte en 64 puntos de imagen.

La altura de los pixel se define en bit 4 de SDMCTL (559 = \$22F). En estado normal, con el bit 4 desactivado, cada pixel tiene una altura de dos líneas de pantalla. Al activar bit 4 en 556 se puede definir la altura en una línea de TV.

Bit tipo	anchura normal	anchura doble
10000001	(=129)	11000000000000011
111100000000000000000000000001111		
01000010	(= 66)	0011000000001100
00001111000000000000000011110000		
00100100	(= 36)	0000110000110000
00000000111100000000111100000000		
00011000	(= 24)	0000001111000000
00000000000011111111000000000000		
00011000	(= 24)	0000001111000000
00000000000011111111000000000000		
00001000	(= 8)	0000000011000000
000000000000000000001111000000000000		
00111000	(= 56)	0000111111000000
00000000111111111111000000000000		
11101111	(=239)	1111110011111111
11111111111100001111111111111111		

(Representación de un jugador en resolución de una línea; cada 1 corresponde a un punto de imagen. En resolución de dos líneas, el tipo de puntos que crea un byte se muestra en dos líneas de pantalla. Los pixel adquieren así altura doble.)

(R) Al ser leído, este registro muestra colisiones entre el proyectil 0 y un jugador.

Bit	7	6	5	4	3	2	1	0
Valor decimal					8	4	2	1
Jugador		no utilizado			3	2	1	0

Cuando el proyectil 0 alcanza al jugador 0, PEEK(53256)
encuentra un 0.

53257-53259 \$\$D009-\$D00B SIZEP1-3

(W) Anchura de jugador 1 a 3.

(R) Colisión entre - proyectil 1 a 3 / jugador.

53260

\$D00C

SIZEM

(W) Cada dos bits de este registro definen la anchura de un proyectil:

Bit	7	6	5	4	3	2	1	0
Valor posicional	128	64	32	16	8	4	2	1
Proyectil	--3--		--2--		--1--		--0--	

Los siguientes pares de bits definen la anchura del proyectil:

- 00 anchura normal, dos puntos de imagen por pixel
- 01 anchura doble, cuatro puntos de imagen por pixel
- 10 anchura normal, dos puntos de imagen por pixel
- 11 anchura cuádruple, ocho puntos de imagen por pixel

Si desea representar proyectil 3 en anchura cuádruple, deberá activar los bits 7 y 6. El bit 4 activado confiere anchura doble al proyectil 2. Si los proyectiles 1 y 0 han de mantener anchura normal, se desactivan los bits 3 a 0. De esta forma se crea el bit tipo 11 01 00 00 cuyos valores posicionales son $128+64+0+16+0+0+0 = 208$. Esto significa que deberá POKEar un 208 después de SIZEM para dar lugar a las mencionadas anchuras de los diversos proyectiles.

(R) PEEK puede investigar aquí si se ha efectuado una colisión entre jugador 0 y otro jugador.

Bit	7	6	5	4	3	2	1	0
Valor decimal					8	4	2	1
Jugador					3	2	1	-

53261

\$D00D

GRAFP0

(W) Aquí se puede definir una forma gráfica para jugador 0. Sin embargo, esto sólo tiene efecto cuando se ha desconectado el DMA de GRACTL (53277 = \$D01D) para jugador; de lo contrario este registro gráfico se cargaría a través del acceso directo a la memoria (DMA) sobre los datos del jugador cargados en la memoria de PM.

Si se desconecta DMA, el BASIC sólo permite colocar aquí un byte cuyo bit tipo determine la forma del jugador en toda su altura, es decir se originan líneas verticales. Al escribir por ejemplo un 255 después de GRAFP0, se activan los ocho bits y el jugador aparece en forma de barra de ocho bits de ancho, o sea de 16 puntos de imagen si la anchura es normal. A causa de esta limitación, la aplicación del registro GRAFPn es muy reducida.

(R) Colisión entre - jugador 1 / jugador.

53262-53263

\$D00E-\$D00F

GRAFP1-2

(W) Forma de jugador 1 y 2.

(R) Colisión entre - jugador 1 / jugador.

53264

\$D010

GRAFP3

(W) Forma del jugador 3.

(R) Flag del botón de fuego 0 del joystick. En BASIC se escribe en él a través del registro de sombra STRIG0 (644 = \$284).

El botón de fuego 0 del joystick ocupa el pín 6 del casquillo 1 del joystick. Si el disparador no está pulsado,

bit 0 tiene valor 1 que será cambiado a 0 al abrir fuego. Al activar bit 2 de GRACTL (53277 = \$D01D), todos los registros TRIGn leen un 0 hasta que GRACTL vuelva a tomar su valor inicial. Con ello se puede regular fuego continuo para todos los botones de fuego.

53265 \$D011 GRAFM

(W) Forma de todos los proyectiles. Trabaja igual que GRAFPn.

Cada par de bits corresponde a un proyectil:

Bit	7	6	5	4	3	2	1	0
Proyectil	--3--	--2--	--1--	--0--				

Cada bit activado crea una línea vertical del ancho de un pixel, que se extiende por toda la pantalla. Naturalmente se puede definir por separado la forma de cada proyectil, activando los bits correspondientes y colocando en GRAFM el valor decimal de este bit tipo.

(R) Botón de fuego 1 del joystick. Registro de sombra STRIG1 (645 = \$285).

53266 \$D012 COLPM0

(W) Color de jugador y proyectil 0. Los proyectiles siempre tienen el mismo color que el jugador que le corresponde. El valor de color de jugador/proyectil 0 se define en BASIC a través del registro de sombra PCOLR0 (704 = \$2C0).

Cuando los cuatro proyectiles (activando el bit 4 de GPRIOR 623 = \$26F) se agrupan para formar un quinto jugador, este último obtiene su color del registro COLOR3 (711 = \$\$2C7), el registro de color del campo de juego 3 (no utilizado en

gran parte de los modos gráficos).

(R) Botón de fuego 2 del joystick. Registro de sombra STRIG2 (646 = \$286).

53267 \$D013 COLPM1

(W) Valor del color de jugador/proyectil 1. Registro de sombra PCOLR1 (705 = \$2C1).

(R) Botón de fuego 3 del joystick. Registro de sombra STRIG3 (647 = \$287).

53268 \$D014 COLPM2

(W) Valor del color de jugador/proyectil 2. Registro de sombra PCOLR2 (706 = \$2C2).

(R) Se utiliza para determinar la compatibilidad del dispositivo con PAL (bits 1 a 3 activados a 0) o NTSC (bits 1 a 3 activados a 1 = decimal 14).

NTSC es la norma de TV usada en Norteamérica, mientras que PAL es la norma utilizada en gran parte de los países europeos, así como por ejemplo en Israel. Una tercera norma estándar, el sistema SECAM, ha sido introducido en Francia, RDA y URSS.

PAL está acoplado al tiempo de 50 hertz; es decir, un VBLANK de PAL se efectúa cada 1/50 segundo, o sea que es un 12% más lento que el NTSC con un 1/60 segundo. Sin embargo, con sus 2,217 MHz la ejecución en la versión de 50 Hz del ATARI es un 25% más rápida que en la versión americana de 60 Hz.

El sistema PAL trabaja además con 312 líneas de pantalla (scan lines) en lugar de 262. Esta diferencia se compensa al

comenzar los display lists de la versión PAL con 24 líneas vacías. Esto quiere decir que con un ATARI de PAL de hecho se puede aprovechar una sección mayor de la pantalla. La forma de hacerlo se podrá leer en el apéndice display list. El efecto de color de la versión PAL está ajustado por hardware.

53269 \$D015 COLPM3

Valor de color del jugador/proyectil 3. Registro de sombra PCOLR3 (707 = \$2C3).

53270-53273 \$D016-\$D019 COLPF0-3

Color de los campos de juego 0 a 3. Registro de sombra COLOR0 a 3 (708-711 = \$2C4-\$2C7).

53274 \$D01A COLBK

Valor de color del fondo (BAK). Registro de sombra COLOR4 (712 = \$2C8).

53275 \$D01B PRIOR

(W) Aquí se determinan los elementos gráficos (jugadores, proyectiles y campos de juego) que ocultan a otros al producirse una superposición. En BASIC, sólo se puede escribir en este registro a través del registro de sombra GPRIOR (623 = \$26F), porque después de PRIOR se escribiría encima de un POKE del BASIC con el próximo tiempo de máquina, mientras que en PRIOR se iría renovando continuamente el valor colocado en el registro de sombra.

Las siguientes prioridades se definen al activar el bit correspondiente (PL=jugador, PF=campo de juego, BAK=fondo):

Bit 3	bit 2	bit 1	bit 0
PF 0	PF 0	P 0	P 0
PF 1	PF 1	P 1	P 1
P 0	PF 2	PF 0	P 2
P 1	PF 3/P 5	PF 1	P 3
P 2	P 0	PF 2	PF 0
P 3	P 1	PF 3/P 5	PF 1
PF 2	P 2	P 2	PF 2
PF 3/P 5	P 3	P 3	PF 3/P 5
BAK	BAK	BAK	BAK

Si se activan prioridades contradictorias en los bits 0 a 3, los elementos gráficos que se encuentran en la zona de superposición devienen negros.

El bit 4 activado reúne los cuatro proyectiles para formar el quinto jugador.

El bit 5 activado opera con OR los valores de color superpuestos.

Los bits 6 y 7 determinan el modo GITA 9 a 11.

(más información en GPRIOR (623 = \$26F))

53276

\$D01C

VDELAY

(W) Posibilita el movimiento aislado de jugadores y proyectiles cuando se ha conectado resolución de dos líneas. Al activar el bit correspondiente, el elemento en cuestión se mueve una línea de TV hacia abajo. Si se activa DMA se da lugar al movimiento en más de una línea al cambiar el bit tipo en la memoria del PM. (vea programa ejemplo PMGRPLAY.BEC).

Bit 7	(=128)	jugador	3
Bit 6	(= 64)	jugador	2
Bit 5	(= 32)	jugador	1
Bit 4	(= 16)	jugador	0
Bit 3	(= 8)	proyectil	3
Bit 2	(= 4)	proyectil	2
Bit 1	(= 2)	proyectil	1
Bit 0	(= 1)	proyectil	0

53277

\$D01D

GRCTL

(W) Se puede escribir en los bits 0 a 2.

Si se activa el bit 2, todos los botones de fuego de joysticks y paddles conectan a fuego continuo al accionar un disparador aislado. Sin embargo es imposible activar un único botón de fuego a fuego continuo. Este no finaliza hasta que el bit 2 vuelva a tomar el valor 0.

Con bit 1 se activan los jugadores y con bit 0 los proyectiles del gráfico de PM.

53278

\$D01E

HITCLR

(W) Cualquier valor escrito en este registro borra todos los registros de colisión del PM. Puesto que un registro de colisión sólo podrá registrar una nueva colisión si ha sido borrado previamente, se aconseja escribir con regularidad en HITCLR.

53279

\$D01F

CONSOL

Flag de las teclas de función OPTION, SELECT y START.

Si no se ha pulsado ninguna tecla los bits 0 a 2 están activados, con lo que CONSOL leerá un 7. La pulsación de un tecla vuelve a colocar el bit correspondiente a 0:

Tecla	bit	estado del bit correspondiente							
OPTION	2	0	0	0	0	1	1	1	1
SELECT	1	0	0	1	1	0	0	1	1
START	0	0	1	0	1	0	1	0	1
Valor decimal		0	1	2	3	4	5	6	7

Los dos siguientes programas ejemplo le mostrarán ahora la forma de programar realmente el gráfico de PM (aunque la explicación detallada de los registros se hará más adelante):

```

0 REM PMGRPLAY.BEC
1 REM *****
2 REM * *
3 REM * ROMANTICO AMANECER *
4 REM * *
5 REM *****
10 GRAPHICS 23
20 POKE 708,226:POKE 709,208:POKE 710,6:POKE 712,128
30 COLOR 1:FOR X=0 TO 5:PLOT X,0:DRAWTO X,95:NEXT X
40 FOR X=152 TO 159:PLOT X,0:DRAWTO X,95:NEXT X
50 FOR Y=0 TO 69:READ R:PLOT 6,Y:DRAWTO R,Y:NEXT Y
60 FOR Y=0 TO 53:READ L:PLOT L,Y:DRAWTO 151,Y:NEXT Y
70 DATA
37,38,39,40,40,41,41,41,40,38,36,35,34,33,33,34,33,35,36,38,3
7,37,36,35,34,28,25,24,23,22,23,24,24

```

```

80 DATA
26,27,26,24,22,20,17,18,19,20,20,18,16,14,11,8,7,8,7,9,8,10,1
3,15,15,14,14,10,9,11,10,10,9,7,8,7
90 DATA
148,146,144,141,138,139,143,147,148,143,141,147,145,143,145,1
43,140,137,134,131,128,129,130,134,138,143,141
100 DATA
139,137,138,140,136,132,128,123,124,125,128,131,135,139,143,1
49,147,145,142,139,135,131,126,121,119
110 DATA 120,122,125,131
120 COLOR 3:Z=Z+1
130 X=INT(RND(0)*80)+40:Y=INT(RND(0)*70):PLOT X,Y
140 IF Z<30 THEN 120
150 COLOR 2:FOR Y=0 TO 7:PLOT 6,81+Y:DRAWTO 52,84+Y:DRAWTO
80,80+Y:DRAWTO 135,86+Y:DRAWTO 150,82+Y:NEXT Y
160 FOR Y=86 TO 95:PLOT 6,Y:DRAWTO 150,Y:NEXT Y
200 POKE 704,34
210 POKE 623,8
220 POKE 559,42
230 P=PEEK(106)-20
240 POKE 54279,P
250 PMBASE=P*256
260 POKE 53256,3
270 POKE 53277,2
280 X=30:Y=91
300 POKE 53248,X
310 FOR J=PMBASE+512 TO PMBASE+639:POKE J,0:NEXT J
320 FOR J=PMBASE+512+Y TO PMBASE+512+31+Y:READ D:POKE
J,D:NEXT J
330 DATA
24,60,60,60,126,126,126,126,126,255,255,255,255,255,255,255
340 DATA
255,255,255,255,255,255,255,126,126,126,126,126,126,60,60,60,
24
400 S=STICK(0)
410 IF S=15 THEN 400
420 IF S=7 THEN X=X+1
430 IF S=11 THEN X=X-1
440 POKE 53248,X
450 IF S=13 THEN GOSUB 500
460 IF S=14 THEN GOSUB 600
470 GOTO 400

```

```

500 ↑F Y>92 THEN RETURN
510 FOR J=32 TO 0 STEP -1
520 POKE PMBASE+512+Y+J,PEEK(PMBASE+511+Y+J+J)
530 NEXT J
540 C=C-1:IF C<0 THEN C=0
550 CO=INT(C/15)*2
560 POKE 704,34+CO:POKE 712,128+CO/2:POKE 709,208+CO
570 Y=Y+1
580 RETURN
600 IF Y<16 THEN RETURN
610 FOR J=0 TO 32
620 POKE PMBASE+511+Y+J,PEEK(PMBASE+512+Y+J)
630 NEXT J
640 C=C+1:IF C>75 THEN C=75
650 CO=INT(C/15)*12
660 POKE 704,34+CO:POKE 712,128+CO/2:POKE 709,208+CO
670 Y=Y-1
680 RETURN

```

10: El comando GRAPHICS se refiere tan sólo al "campo de juego", mientras que los elementos de PM son completamente independientes del mismo.

20: Se colocan los valores de color de los puntos gráficos ("campos de juego") y del fondo en sus correspondientes registros de color.

30 a 110: Dibujan de COLOR 3 la silueta de un árbol de fronda a la izquierda y un abeto en el margen derecho de la pantalla.

120 a 140: Dibujan (PLOT) aleatoriamente treinta estrellas en el cielo nocturno.

150 a 160: Plasman una cordillera pintoresca en la pantalla de TV.

200: Ahora comienza gráfico de PM. El color del jugador 0 se coloca en el correspondiente registro de color.

210: La prioridad se define de tal forma que montañas y árboles se encuentren por delante del jugador 0; el fondo representa al cielo oscuro, que se encuentra por detrás del jugador 0.

220: Se conectan la anchura normal del campo de juego (bit 1 = 2), el jugador (bit 3 = 8) y el acceso directo a la memoria (bit 5 = 32).

230: En RAMTOP se averigua dónde se encuentra el límite de la memoria. GRAPHICS 23 ocupa casi 16 páginas con su display list. Puesto que el gráfico de PM debe trabajar en resolución de dos líneas, basta aquí con 4 páginas. Esto significa que la dirección básica del gráfico de PM debe encontrarse unas 20 páginas por debajo de RAMTOP. Este valor se define en P.

Si cuenta aquí como prueba menos páginas y resta por ejemplo un 8 de RAMTOP, más adelante podrá ver en el gráfico de pantalla, de qué forma el jugador se encuentra en la memoria de pantalla y se mueve en ella.

240: El registro 54279 es PMBASE, dirección inicial de la memoria de PM, Puesto que PMBASE trata sólo con el byte HI al igual que RAMTOP, la verdadera

250: dirección inicial se obtiene al multiplicar el valor por 256.

260: El jugador 0 deviene de anchura cuádruple. Cada uno de los ocho bits es representado con una anchura de ocho puntos de imagen; el jugador ocupa un total de 64 puntos de imagen en la pantalla.

270: Conecta jugador.

280: Coordenadas iniciales del jugador.

300: En HPOSP0 se coloca $X=30$ como posición horizontal del jugador 0. Esta posición corresponde al borde extremo (visible) de la pantalla de TV. Al desplazar el jugador aún más hacia la izquierda, éste desaparecerá de la pantalla, pero no se produce una interrupción por ERROR- hasta que el valor de X no sea inferior a 0, o el del lado derecho no sea mayor que 255.

310: Las posiciones de memoria 512 a 639 después de PMBASE están reservados para jugador 0 con una resolución de dos líneas. Aquí se borra esta zona de la memoria, aunque ello no es absolutamente necesario. Sin embargo, una interrupción del programa, provocada por ejemplo con BREAK, no borra esta memoria de PM; así puede ocurrir que, al iniciar de nuevo el programa, aparezcan dos jugadores en la pantalla. Uno de ellos, el inmóvil, se encuentra en la posición vertical donde se produjo también la interrupción del programa, mientras que el segundo jugador se encuentra en el lugar hacia el cual ha sido cargado por el programa que vuelve a ejecutarse, o sea en su posición inicial.

Si ejecuta este programa, lo interrumpe con BREAK y después vuelve a entrar RUN, puede observar que el jugador aparece primero en la antigua posición vertical, a continuación es borrado por esta línea y finalmente es plantado en su posición de salida.

Sería, pues más correcto borrar en primer lugar la memoria de PM y conectar después los jugadores en el registro 53277.

320: Los bytes de datos que representan al jugador son leídos por la zona de memoria correspondiente al jugador 0. Un jugador siempre tiene un ancho de un byte y, con resolución de dos líneas, una altura de 128 bytes. Estos 128 bytes disponen de las celdas de memoria $PMBASE+512$ a $PMBASE+639$. El primer byte será leído en la celda de memoria situada $Y=91$ (valor de la posición vertical) por debajo de $PMBASE+512$ y le suceden los 31 bytes siguientes.

330 y 340: Contienen los DATA que representan al jugador. Cada valor decimal representa un bit tipo. Cada bit activado crea un píxel cuya anchura ha sido definida en el registro correspondiente (en este caso cuádruple = ocho puntos de imagen), y cuya altura es de dos líneas de pantalla en resolución de dos líneas, y de una en resolución de una línea, la cual está definida en el registro 559 al activar o desactivar el bit 4.

400: En esta línea comienza la lectura del joystick. Es aconsejable recoger el valor hallado en STICK0 (632 = \$278) en una variable, para evitar el uso reiterado de la larga expresión STICK(0). Por cierto, el comando BASIC STICK(0) no tiene otro efecto que la instrucción PEEK(632), pero STICK(0) ocupa menos memoria.

410: Cuando STICK(0) está en reposo (=15) vuelve a ser leído inmediatamente.

420: La palanca de mando está dirigida a la derecha, con lo que el jugador deberá moverse hacia la derecha, o sea que su posición horizontal deberá tomar mayor valor.

430: Movimiento correspondiente hacia la izquierda.

440: La posición horizontal del jugador es escrita simplemente en el registro que le corresponde. Este proceso es fácil y rápido. Los jugadores se mueven con bastante rapidez en el sentido horizontal.

450: No así el movimiento vertical. Un jugador se mueve en sentido vertical al trasladar sus datos en su zona de memoria. Si el jugador debe moverse hacia abajo, todos sus bytes de datos también deben trasladarse una celda de memoria hacia abajo. Cuantos más datos abarque el jugador, más tiempo tardará el proceso. Usted puede observar en la pantalla cómo el jugador va descendiendo línea por línea. La rutina encargada de ello está colocada en el subprograma de la línea 500.

460: El movimiento ascendente se efectúa de forma análoga.

470: Después de leer todas las direcciones del movimiento, el programa es devuelto a la siguiente lectura del joystick.

500: Aquí comienza la rutina que controla el movimiento descendente. Sin embargo, al alcanzar la posición Y=92, debería impedirse su continuación. Una lectura de este tipo es absolutamente necesaria, ya que en caso contrario los datos van trasladándose desenfrenadamente por zonas de la memoria a las que no deberían acceder, como por ejemplo por sectores destinados a otro jugador.

510: Puesto que en este caso el jugador tiene una altura de 32 bytes, hay que trasladar 33 (de 0 a 32) bytes. En el movimiento descendente se comienza por el byte más bajo que se coloca una dirección más abajo; a continuación se coge el byte inmediatamente superior que también se coloca hacia abajo, etc. Si pone atención, podrá apreciar cómo el jugador se estira por un momento en cada línea al moverse hacia la línea inferior. Y tal como suena este proceso, así es su duración en BASIC.

Después de trasladar todos los bytes hacia posiciones inferiores, habrá que trasladar además otro byte situado en posición superior. Este byte contiene un 0 porque el jugador finaliza con él. De esta forma se borra el último byte de su antigua celda después de haberlo trasladado hacia abajo.

540 a 560: Cuando en este romántico programa, el sol-jugador 0- sale lentamente detrás de las montañas de COLOR-2 y proyecta su luz roja a través de los árboles de COLOR-1, también debería clarear. El color del sol pasa lentamente de naranja a blanco, el cielo deviene de color azul claro apagando las estrellas de RANDOM, y los prados relucen en un verde que transmite la sensación de frescura. ¡De esta forma puede que incluso la abuelita se entusiasme por el ordenador!

Estas líneas proceden a cambiar los valores de color cuando el jugador (sol) va descendiendo, o sea con la puesta de sol; los colores devienen más oscuros y el valor de claridad disminuye.

570: Ahora sólo resta actualizar el valor de Y y volver con

580: RETURN al programa principal.

600 a 680: Su estructura es análoga para realizar el movimiento ascendente.

La forma de trabajar con los registros de colisión se muestra en el player/missile-lightshow:

```
0 REM PMGRCOLL.BEC
1 REM *****
2 REM * *
3 REM * PLAYER/MISSILE LIGHT-SHOW *
4 REM * *
5 REM *****
10 GRAPHICS 19
200 POKE 704,50:POKE 705,152
210 POKE 623,8+32
220 POKE 559,42
230 P=PEEK(106)-8
240 POKE 54279,P
250 PMBASE=P*256
260 POKE 53256,3:POKE 53257,3
270 POKE 53277,2
280 X0=30:Y0=92:X1=190:Y1=16
300 POKE 53248,X0:POKE 53249,X1
```

```

310 FOR J=PMBASE+512 TO PMBASE+767:POKE
J,0:NEXT J
320 FOR J=PMBASE+512+Y0 TO
PMBASE+512+31+Y0:READ D:POKE J,D:NEXT
J:RESTORE
330 FOR J=PMBASE+640+Y1 TO
PMBASE+640+31+Y1:READ D:POKE J,D:NEXT J
340 DATA
24,60,60,60,126,126,126,126,126,255,255,2
55,255,255,255,255
350 DATA
255,255,255,255,255,255,255,126,126,126,1
26,126,60,60,60,24
400 S=STICK(0)
410 IF S=15 THEN 700
420 IF S=7 THEN X0=X0+1
430 IF S=11 THEN X0=X0-1
440 POKE 53248,X0
450 IF S=13 THEN GOSUB 500
460 IF S=14 THEN GOSUB 600
470 IF PEEK(53260)=2 THEN GOSUB 1000
480 GOTO 700
500 IF Y0>92 THEN RETURN
510 FOR J=32 TO 0 STEP -1
520 POKE
PMBASE+512+Y0+J,PEEK(PMBASE+511+Y0+J)
530 NEXT J
540 Y0=Y0+1:RETURN
600 IF Y0<16 THEN RETURN
610 FOR J=0 TO 32
620 POKE
PMBASE+511+Y0+J,PEEK(PMBASE+512+Y0+J)
630 NEXT J
640 Y0=Y0-1:RETURN
700 S=STICK(1)
710 IF S=15 THEN 400
720 IF S=7 THEN X1=X1+1
730 IF S=11 THEN X1=X1-1
740 POKE 53249,X1
750 IF S=13 THEN GOSUB 800
760 IF S=14 THEN GOSUB 900
770 IF PEEK(53261)=1 THEN GOSUB 1000
800 IF Y1>92 THEN RETURN
810 FOR J=32 TO 0 STEP -1
820 POKE
PMBASE+640+Y1+J,PEEK(PMBASE+639+Y1+3)
830 NEXT J
840 Y1=Y1+1:RETURN
900 IF Y1<16 THEN RETURN
910 FOR J=0 TO 32
920 POKE
PMBASE+639+Y1+J,PEEK(PMBASE+640+Y1+J)
930 NEXT J
940 Y1=Y1-1:RETURN
1000 SOUND 0,3,6,8
1010 FOR Q=0 TO 6:POKE 712,Q*2:NEXT Q
1020 POKE 53278,255
1030 POKE 712,0
1040 SOUND 0,0,0,0
1050 RETURN

```

210: Además de la prioridad de jugadores y campos de juego, en este caso también se activa el tercer color resultante de la operación OR al producirse la superposición (+32). Por supuesto podrá POKEar Inmediatamente un 40.

220: DMA Igual que antes.

230: En este caso, gracias a la reducida ocupación de memoria del modo gráfico 3 (o sea 19), la base de PM sólo debe encontrarse a ocho páginas debajo del RAMTOP.

260: Ambos jugadores adquieren anchura cuádruple.

280: Jugador 0 comienza en la parte inferior izquierda y jugador 1 en la superior derecha.

310: Esta vez se borra la memoria correspondiente a los jugadores 0 y 1; a continuación la línea

320: lee los datos del jugador 0 y la línea

330: los datos del jugador 1. Puesto que ambos han de tener la misma configuración, después de RESTORE el jugador 1 lee en línea 330 los mismos

340 y 350: DATA que el jugador 0.

400 a 460: Efectúan la lectura de STICK(0) y saltan hacia la línea 700, donde comienza el turno de STICK(1).

470: SI PEEK(53260)=2, el jugador 0 ha tropezado con jugador 1. Se produce un salto hacia la subrutina en línea 1000.

500 a 640: Se encargan de efectuar el movimiento vertical de jugador 0.

700 a 760: Leen a STICK(1).

770: SI se lee un 1 en el registro 53261, significa que el

jugador 1 ha tropezado con jugador 0.

800 a 940: Realizan los traslados verticales de jugador 1.

1000: Si permite que los dos jugadores se lancen uno sobre otro, esto se convierte en un infierno.

Pero eso tendrá que observarlo usted mismo.

1020: En ningún caso debemos olvidar de volver a borrar inmediatamente los registros de colisión, para que éstas pueden continuar produciéndose.

Los dos jugadores circulares tienen aspecto de faros. Jugador 0 luce un rojo bastante puro y jugador 1 brilla casi en cyan (azul DIN). Cuando ambos se superponen, la superficie de intersección adquiere un color amarillo verdoso, lo que corresponde casi a la realidad física. (De luz proyectada roja y verde resulta una mezcla de color amarillo.)

Resumiendo el cálculo del color de cruce:

Rojo (color 3, claridad 2) 0 0 1 1 0 0 1 0 (= 3*16 + 2 = 50)

Cyan (color 9, claridad 8) 1 0 0 1 1 0 0 0 (= 9*16 + 8 = 152)

Verde (color 11, claridad 10) 1 0 1 1 1 0 1 0 (= 11*16 + 10 = 186)

Por cierto, si interrumpe un programa PM con BREAK, es normal que en lugar del jugador o jugadores permanezcan cintas centelleantes. Suele bastar un RESET para eliminar este efecto molesto.

SONIDO

53760 \$D200 AUDF1

El ATARI esta equipado con cuatro generadores de sonido controlados por el POKEY (POtenciómetro y chip KEYboard). A cada generador de sonido le corresponde un registro para la frecuencia y un registro de control destinado a la intensidad y distorsión del sonido. La distorsión es producida por los polycounter y registros de desplazamiento de POKEY.

(W) AUDF1 recoge la frecuencia del canal de sonido 1. Pueden escribirse valores de 0 a 255. POKEY les suma un 1, con lo que se activan valores entre 1 y 256. Este valor determina la cantidad de impulsos que han de entrar para poder emitir un impulso. Esto significa que cuanto mayor sea el valor colocado en este registro, menor será la cantidad de impulsos emitidos. Nosotros percibimos una frecuencia baja como sonido grave.

Si sabe más de música que lo de tener que soplar para hacer sonar una flauta, podrá determinar el valor exacto a colocar en este registro para conseguir un cierto sonido. La fórmula es la siguiente:

$$\text{INT}(31960/(f+2))$$

donde hay que sustituir f por la frecuencia en hertz (Hz) del sonido deseado.

(R) Valor de paddle (raquetas) 0. Registro de sombra 624.

53761 \$D201 AUDC1

(W) Los tres bits superiores se utilizan para determinar el

tipo de distorsión (ruido):

Bit			Valor de distorsión	Proceso de modulación
7	6	5	(BASIC)	(POKEY)
0	0	0	0	0
polycounter de 17 bits				
0	0	1	2	sólo polycounter de 5 bits
0		1	0	4
polycounter de 4 bits				
1	0	0	6	sólo polycounter de 17 bits
0	1	1	8	sólo polycounter de 5 bits
1	0	1	10	sin distorsión
1	1	0	12	sólo polycounter de 4 bits
1	1	1	14	sin distorsión

Los bits 0 a 3 toman el valor decimal de la intensidad de sonido. Se admiten valores de 0 (0000) a 15 (1111).

El comando BASIC SOUND k,f,r v se dirige con k a uno de los canales de sonido de 0 a 3. El número de identificación f determina la frecuencia en el registro AUDFn; r determina la distorsión (valores pares de 0 a 14) y v la intensidad del sonido entre 0 y 15 en el registro AUDCn.

(R) Paddle 1.

53762 \$D202 AUDF2

(W) Frecuencia del canal de sonido 2.

(R) Paddle 2.

53763 \$D203 AUDC2

(W) Registro de control correspondiente al canal de sonido 2.

(R) Paddle 3.

53764 \$D204 AUDF3

(W) Frecuencia del canal de sonido 3.

(R) Paddle 4.

53765 \$D205 AUDC3

(W) Registro de control correspondiente al canal de sonido 3.

(R) Paddle 5.

53766 \$D206 AUDF4

(W) Frecuencia del canal de sonido 4.

(R) Paddle 6.

53767 \$D207 AUDC4

(W) Registro de control correspondiente al canal de sonido 4.

(R) Paddle 7.

(W) Control de sonido. AUDCTL es un registro de selección que influye en todos los canales de sonido. Al activar los diversos bits se puede dar lugar a los efectos siguientes:

Bit	Efecto
7	polycounter de 17 bits se pone a 9 bits
6	canal 1 es indexado a 2,217 MHz
5	canal 3 es indexado a 2,217 MHz
4	comunica los canales 1 y 2 (16 bits)
3	comunica los canales 3 y 4 (16 bits)
2	filtro de tonos agudos en canal 1, indexado por canal 4
1	filtro de tonos agudos en canal 2, indexado por canal 4
0	conmuta el ciclo principal de 64 kHz a 15 kHz

(R) Lee todos los paddles al mismo tiempo, cada uno de ellos en un bit.

VARIADO

53770 \$D20A RANDOM

(R) En este registro se leen los valores aleatorios. Contiene los ocho bits superiores del polycounter de 17 bits. De esta forma, aquí se encuentran continuamente verdaderos números aleatorios entre 0 y 255. El comando BASIC RND(0) también hace referencia a este registro, pero convierte el número aleatorio hallado en un número decimal de valor positivo y menor que 1, dividiendo el valor de RANDOM entre 256. De esta forma, las instrucciones PRINT RND(0) y PRINT PEEK(53770)/256 tienen el mismo resultado.

Para obtener números aleatorios enteros entre 5 y 14, se programa en BASIC INT(RND(0)*10)+5, donde RND(0)*10 nos da un valor entre 0 y 9,99... que INT convierte en números enteros (de 0 a 9). Al sumarle 5, se obtienen los valores deseados entre 5 y 14. INT(PEEK(53770)/25.6)+5 produce el mismo resultado.

53774 \$D20E IRQEN

(W) Restitución de interrupción. Registro de sombra POKMSK (16 = \$10); vea más información en ese registro.

54272 \$D400 DMACTL

Registro de control de DMA. En BASIC se escribe en él a través del registro de sombra SDMCTL (559 = \$22F). (vea más información allí)

54273 \$D401 CHACTL

Registro de control del modo carácter. En él se escribe a través del registro de sombra CHACT (755 = \$2F3). (vea más información en ese registro)

54274,54275 \$D402,\$D403 DLISTL/H

Bytes LO y HI del puntero que apunta hacia el display list. (vea apéndice display list).

54279 \$D407 PMBASE

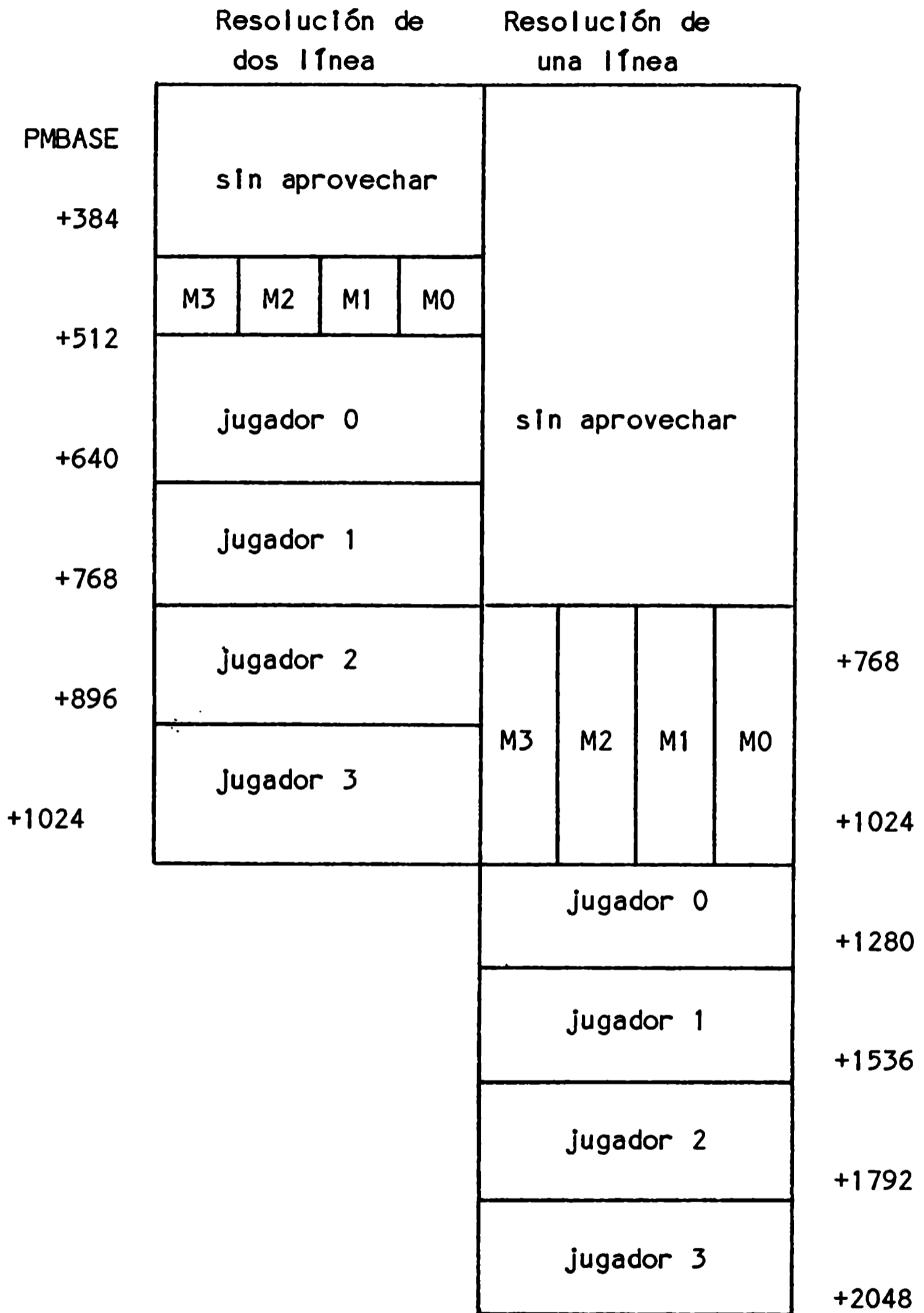
(W) Byte HI de la dirección inicial de PM.

Existe un plano de memoria para los datos de jugadores y proyectiles, que asigna una posición de memoria propia a cada elemento. Un player siempre tiene un byte de ancho y se extiende por toda la pantalla en sentido vertical, saliéndose incluso de la ventana gráfica manejada por los modos gráficos. En resolución de dos líneas un jugador tiene una altura de 128 bytes; en resolución de una línea necesitará el doble, es decir 256 bytes de memoria.

Los cuatro proyectiles que se pueden agrupar para formar un quinto jugador, ocupan juntos la misma memoria que un jugador solo. De ello, junto con una zona de protección no utilizada al principio de la memoria de PM, resulta un total de memoria necesaria de un kbyte en resolución de dos líneas, y de 2 kbyte en resolución de una línea.

La distribución habitual de la memoria de este registro se muestra en la figura 5:

PLANO DE MEMORIA - PM



Es absolutamente necesario respetar esta disposición, puesto que funciones tales como los registros de colisión etc. se apoyan en PMBASE y en esta disposición. Es aconsejable colocar la tabla de PM al final de la memoria RAM más alta, o sea directamente delante del display list y la memoria de pantalla. Sin embargo no existe ningún impedimento para utilizar cualquier otra zona (protegida). Sea cual sea la página (byte HI de la dirección) donde comienza la memoria de PM, hay que depositar este valor en PMBASE.

Para encontrar una dirección inicial de gráfico PM segura y protegida pueden realizarse las siguientes operaciones: redondear por exceso la memoria necesaria del modo gráfico activado y de su correspondiente display list a páginas más la longitud de la tabla de PM, o bien restar 4 o 8 páginas de RAMTOP.

La lectura del puntero hacia el display list SDLSTL (560,561 = \$230,\$231) efectuada después de conectar el modo gráfico deseado conduce al mismo resultado. Esta dirección, redondeada por exceso a páginas completas y restándole la memoria necesaria de PM, también nos da la dirección inicial de PB, cuyo byte HI debe ser colocado en PMBASE.

Partiendo de PMBASE podrá calcular el valor de APPMHI (14,14 = \$E,\$F), sumando a PMBASE la memoria necesaria de cuatro páginas en resolución de dos líneas, y de ocho páginas en resolución de una línea. APPMHI marca el límite inferior del display list y de la memoria de pantalla.

54281

\$D409

CHBASE

(W) Dirección inicial del juego de caracteres estándar del ATARI, almacenado naturalmente en la memoria RAM. El juego está compuesto por 128 caracteres de ocho bits cada uno, ocupando justamente cuatro páginas o un kbyte. En el registro de sombra CHBAS (756 = \$2F4) el programador de

BASIC puede cambiar el puntero que apunta hacia el juego de caracteres. (vea registro de sombra CHBAS)

En el apéndice juego de caracteres podrá encontrar más indicaciones sobre la estructura del juego estándar y la posibilidad de aplicar los juegos de definición propia.

DISPLAY LIST

El display list (DL) es un programa en lenguaje máquina del microprocesador ANTIC. Este programa determina de qué zona de memoria deben sacarse los datos gráficos y de qué forma se representarán en la pantalla. Cuando en BASIC se entra un comando GRAPHICS, con ello se reserva automáticamente la memoria necesaria para la pantalla y delante de esta memoria se coloca el DL correspondiente. Sin embargo, cuando se conoce la estructura del DL y el juego de instrucciones de ANTIC, es posible modificar el DL o confeccionar un DL enteramente personal, en el que pueden mezclarse todas las posibilidades de representación gráfica de ANTIC.

La estructura de un DL puede verse en la pantalla con ayuda del programa ADR560.BEC, que ya se ha discutido en SDLSTL (560,561 = \$230,\$231).

ANTIC procesa las siguientes instrucciones de 8 bits:

1 JMP, comando de salto incondicionado. Inmediatamente detrás de este comando debe seguir la dirección del salto en el orden byte LO y HI. Esta instrucción se utiliza por ejemplo para saltar hacia una subrutina desde el DL, como puede ser un segundo y DL que emite varias líneas gráficas por encima o por debajo de la ventana gráfica normal, y que trabaja independientemente del DL principal.

2 Modo carácter, llamado en BASIC a través de GRAPHICS. Una línea de modo está dividida en 40 posiciones de escritura y tiene una altura de ocho líneas de TV. Se permite representar dos colores al mismo tiempo.

3 Modo carácter, 40 posiciones de escritura, 10 scan lines, 2 colores.

4 Modo carácter (GRAPHICS 12), 40 posiciones de escritura, 8 líneas, 5 colores.

5 Modo carácter (GRAPHICS 13), 40 posiciones de escritura, 16 líneas, 5 colores.

6 Modo carácter (GRAPHICS 1), 20 posiciones de escritura, 8 scan lines, 5 colores.

7 Modo carácter (GRAPHICS 2), 20 posiciones de escritura, 16 líneas, 5 colores.

8 Modo carácter (GRAPHICS 3), 40 pixel por línea, cada línea de modo con altura de 8 líneas de TV, 4 valores de COLOR.

9 Modo bit-map (GRAPHICS 4), 80 pixel, 4 scan lines, 2 valores de COLOR.

10 Modo bit-map (GRAPHICS 5), 80 pixel, 4 líneas, 4 valores de COLOR.

11 Modo bit-map (GRAPHICS 6), 160 pixel, 2 líneas, 2 valores de COLOR.

12 Modo bit-map (GRAPHICS 14), 160 pixel, 1 línea, 2 valores de COLOR.

13 Modo bit-map (GRAPHICS 7), 160 pixel, 2 líneas, 4 valores de COLOR.

14 Modo bit-map (GRAPHICS 15), 160 pixel, 1 línea, 4 valores de COLOR.

15 Modo bit-map (GRAPHICS 8), 320 pixel, 1 scan line, 2 valores de COLOR.

(La indicación de pixel por línea hace referencia a la anchura normal de pantalla)

- 0 una línea de TV vacía.
- 16 dos líneas de TV vacías.
- 32 tres líneas de TV vacías.
- 48 cuatro líneas de TV vacías.
- 64 cinco líneas de TV vacías.
- 80 seis líneas de TV vacías.
- 96 siete líneas de TV vacías.
- 112 ocho líneas de TV vacías.

64 activado además del comando líneas de modo (o sea bit 6 activado)

LMS (load memory scan), es decir los datos gráficos son leídos en la memoria de pantalla. Cada instrucción LMS debe ir seguida de la dirección inicial de los datos de pantalla en forma de bytes LO y HI.

65 JVB, salto condicionado por el VBLANK. Con esta instrucción situada al final del DL se consigue la sincronización entre ordenador y tubo de televisión. Una vez llegado al final del DL, el ordenador espera la señal del VBLANK hasta que vuelve a saltar al principio del DL, empezando así una nueva pasada sincronizado con el tubo de TV. Al igual que en todas las instrucciones de salto, también JVB debe ir seguido directamente de la dirección de salto (LO,HI).

128 DLI (display list interrupt), sumado al comando de líneas de modo, el valor 128 (bit 7) posibilita la interrupción del DL para procesar en HBLANK una corta rutina en lenguaje máquina. Vea el programa ejemplo ADR512.BEC.

Los display lists estándar manejan 192 scan lines. Estas van precedidas de tres veces ocho líneas vacías para compensar el sistema en la versión PAL, de manera que se manejan exactamente $192+24$, o sea 216 líneas de TV. Cambiando el DL, estas 24 líneas vacías pueden utilizarse activamente. Un DL modificado también puede trabajar con menos de 216 o 192 scan lines, apareciendo una ancha franja vacía (negra) en la parte inferior de la pantalla. El comando JVB obliga al ordenador a esperar al haz catódico del tubo de televisión, de manera que cualquier DL más corto pueda ir sincronizado con el monitor.

Puesto que el sistema PAL divide el tubo de televisión en 312- en lugar de las 262 líneas del sistema NTSC- las 24 líneas vacías al principio del DL provocan que ventana gráfica de la pantalla PAL se encuentre, aproximadamente, en el centro vertical del tubo de TV. Sin embargo, esto significa que debajo del DL estándar también pueden aprovecharse unas 20 líneas adicionales. Pero si la longitud del DL excede del número de líneas con que trabaja el televisor y el DL todavía no se ha procesado por completo cuando el tubo de TV atraviese el VBLANK, será imposible conseguir la sincronización, la imagen se distorsiona y empieza a correr. Sin embargo no se producen defectos en el tubo de televisión.

El programas siguiente muestra la forma de programar un display list individual en BASIC:

```

0 REM DLTWUP.BEC
1 REM *****
2 REM *                               *
3 REM *   GRAPHICS 8. VENTANAS       *
4 REM *   SUPERIORES DE TEXTO        *
5 REM *                               *
6 REM *****
10 GRAPHICS 24:POKE 703,4
20 DL=PEEK(560)+PEEK(561)*256
30 FOR J=0 TO 9:READ B:POKE DL+J,B:NEXT J
40 DATA 112,66,96,159,2,2,2,79,80,129
100 COLOR 1
110 PLOT 0,0:DRAWTO 319,191
120 PLOT 319,0:DRAWTO 0,191
130 PLOT 0,0:DRAWTO 100,89
140 PLOT 100,93:DRAWTO 207,191
200 PRINT "GRAPHICS 8,VENTANAS SUPERIORES DE TEXTO"

```

10: Tanto si sólo modifica ligeramente un display list estándar como si construye uno completamente nuevo, en BASIC no podrá evitar llamar a un modo gráfico, puesto que el comando GRAPHICS realiza algunas tareas que no pueden sustituirse fácilmente ni siquiera por PEEKs y POKEs.

Un función importante consiste en que GRAPHICS reserve memoria en la medida que el modo en cuestión precise para memorizar los datos de imagen. Si usted confecciona un DL propio tendrá que calcular la necesidad de SM y conectar un modo gráfico estándar que tenga, al menos, igual o mayor necesidad de memoria. En una tabla del apéndice memoria de pantalla puede encontrar la memoria necesaria de las diversas líneas de modo.

Al llamar un modo gráfico con ayuda del comando GRAPHICS, también se coloca un display list correspondiente en la memoria a partir del lugar marcado por el vector SAVMSC (88,89 = \$58,\$59). Si el OS se encarga de estas cosas, no hay problemas; pero si uno empieza a experimentar con sus propios programas, tendrá que atenerse a las consecuencias, puesto que un display list no debe superar el límite de 1k sin ejecutar una nueva instrucción de salto (JMP, etc.). El DL más largo, el de GRAPHICS 8, no excede de 202 bytes. De esta forma no existe ninguna dificultad en emplazar un DL entre límites de 1k. Pero hay que tenerlo en cuenta.

A pesar de ello un DL puede alcanzar una longitud considerable. La instrucción LMS de ANTIC activa el bit 6 (= 64) del byte que contiene la instrucción para una línea de modo, es decir, desde el punto de vista decimal, suma 64 al número de identificación de una línea de modo. En teoría es posible dotar cada una de las líneas de modo con un LMS, asignando de esta forma una zona de memoria propia a cada línea gráfica. Sin embargo, cada instrucción LMS debe ir seguida del vector que apunta hacia la memoria de pantalla, lo que significa añadir dos bytes adicionales. Cuanto mayor sea la longitud de DL, mayor es el peligro de superar el límite de 1k.

Una segunda restricción consiste en que la memoria que contiene los datos gráficos (SM) no debe superar un límite de 4k sin efectuar una instrucción LMS en el DL. Si observa los DL que ocupan más memoria, los de GRAPHICS 8 a 11, 14 y 15 (necesidad de SM 7680 bytes), algo más abajo del centro encontrará una nueva instrucción LMS con su correspondiente puntero hacia la dirección que señala algo más abajo del centro de la memoria de pantalla, donde continúan los datos gráficos que hacen referencia a la parte inferior de la pantalla.

El modo gráfico escogido también determina la forma de procesar los datos del SM, o sea el bit tipo DMASK 672 = \$2A0) utilizado, es decir la cantidad de bits por pixel leídos y la de registros de color empleados. Se puede influir en esta función colocando en DINDEX (87 = \$57) un valor que simule un modo gráfico diferente al programado con GRAPHICS.

Sin embargo, GRAPHICS o el otro valor colocado en DINDEX influyen, por su parte, en todos los comandos que hacen referencia a la pantalla como son PLOT, LOCATE, etc.

Aquí, en la línea 10 del programa, se fija además la altura de la ventana de texto en 4 (líneas de GRAPHICS 0), puesto que se ha programado GRAPHICS 24, o sea 8 sin ventana de texto, simplemente porque su DL correspondiente es adecuado a las pretensiones de este programa. La ventana de texto debe ser trasladada hacia el borde superior de la pantalla, por lo que se escoge el DL sin ventana de texto, que no precisará de más modificaciones de su extremo inferior.

20: Calcula la dirección inicial del DL.

30: Comienzan a efectuarse los cambios. El bucle FOR-NEXT lee diez valores y los POKEa en el DL.

40: Aquí se encuentran los valores. El 112 origina 8 líneas de TV vacías. El 66 resulta de LMS (64) + una línea de GRAPHICS 0 (2). Por lo tanto, el nuevo DL comienza 16 líneas de TV normalmente vacías más arriba. 96 y 159 corresponden a LO y HI de los vectores que apuntan hacia la memoria de la ventana de texto. Les siguen otras tres líneas del modo gráfico 1. 79 contiene otra vez una instrucción LMS (64) + el valor de comando 15 para una línea de GRAPHICS 8, seguido de los bytes LO (80) y HI (128) del puntero de SAVMSC de SM. El DL restante puede permanecer invariable.

Eso es todo. Si ahora entra un RUNAM, el bonito mundo nuevo de DL funciona. En el extremo superior de la pantalla aparecerá READY y usted podrá hacer allí todas las cosas divertidas que pueden hacerse normalmente en la ventana de texto.

Sin embargo, si se atreve a introducirse en la zona de GRAPHICS 8, no podrá evitar problemas. Se ha escogido conscientemente GRAPHICS 8 para este programa de demostración, porque en los modos elevados aparecen problemas especiales con los límites de k ya mencionados anteriormente.

La nueva zona de GRAPHICS 8 comienza directamente debajo de la ventana de texto y funciona en su parte superior con todas sus características incluso PLOT. Sin embargo, después llega al límite en el cual figura en el DL la nueva instrucción con el vector que apunta hacia la SM restante. Ahora, sin embargo, la nueva ventana gráfica ya no tiene la altura de 192 líneas de modo (;GRAPHICS 24!) tal como lo espera el OS y lo presupone para todos los PLOTs, sino sólo 176 líneas. Y en algún lugar debe hallarse la diferencia de 16 líneas de modo.

100 y 120: PLOT y DRAWTO trazan aquí una cruz diagonal que se extiende por toda la ventana gráfica, y podrá apreciar que le falta una franja en el centro. Este defecto no puede subsanarse tampoco cambiando el puntero después de la segunda instrucción LMS, puesto que el OS calcula los PLOTS exigidos por el comando DRAWTO sobre la base de que GRAPHICS 8 (o bien 24) está conectado de forma habitual.

130 y 140: En este caso no hay más solución que la de programar todas las instrucciones de pantalla por separado para las dos partes.

200: Solamente la ventana de texto funciona sin ningún problema.

El problemático DL aquí presentado sirve de ejemplo también para demostrar que el camino que parece más simple a primera vista no siempre es el mejor. Sin embargo, si en línea 10 llama GRAPHICS 8 en lugar de GRAPHICS 24 y modifica DL de tal forma, que elimina la ventana de texto del extremo inferior y crea otra en la parte superior, sin convertir líneas de TV vacías en activas que por su parte ocuparán SM, entonces la memoria volverá a coincidir y no habrá ningún tipo de problemas. ¡Pruébalo!

El segundo programa ejemplo establece una zona en el centro de la pantalla de GRAPHICS 0, en la cual la letra aparece muy aumentada, como vista con lupa:

El aumento de la letra se consigue al introducir líneas de GRAPHICS 2. Los caracteres de GRAPHICS 2 se representan de doble ancho y una línea de modo tiene altura doble que las de GRAPHICS 0, es decir 16 scan lines. Las dos nuevas líneas ocupan, pues, en la pantalla, el espacio de cuatro líneas de GRAPHICS 0.

De nuevo se transforman 16 líneas inactivas del extremo superior, trasladando pues el DL de GRAPHICS 0 hacia arriba, para finalizar introduciendo las dos nuevas líneas de modo en las líneas 80 y 90.

Al ejecutar este programa la instrucción GRAPHICS de línea 10 limpia la pantalla. Sin embargo podrá observar el cambio inmediatamente, porque las líneas del modo gráfico 2 tienen un fondo negro.

¡Ahora entre LIST! El modo de editor (GRAPHICS 0) es activado. Pueden efectuarse todas las entradas como es habitual. Cuando el listado aparezca en la pantalla verá la línea 30 en bonitas letras grandes de color naranja sobre fondo negro. También puede programar en esta línea destacada. Vaya con el cursor hacia la línea 30. ¡Oh! ¿Dónde está el cursor ahora?

El cursor se origina con ayuda de una función que modifica en 128 el valor del carácter que se encuentra bajo el cursor, invirtiendo de esta forma los caracteres. EN GRAPHICS 2 no existen caracteres inversos. En lugar de ello se cambia el valor del color.

Bien. Ahora lleve el cursor de letra azul hasta el signo de dos puntos y bórrelo junto con el inútil REM que le sigue. No olvide pulsar RETURN, ya que estamos entrando una nueva línea BASIC. Si ahora vuelve a listar el programa, la línea 30 ya está corregida.

Es posible que haya descubierto algún carácter oscuro en el extremo inferior de la pantalla, pero no se preocupe, no ha cometido ningún error. Lo que sucede es que la memoria reservada con ayuda del comando GRAPHICS no es suficiente para este DL. Falta memoria justo para la última línea.

Aunque el DL continúe leyendo como siempre el contenido de las celdas de memoria, ya se encuentra en una zona que no pertenece a SM. Los datos de estas celdas se convierten igualmente en caracteres, pero sin sentido alguno. Tampoco podrá conducir el cursor hacia esta línea. El editor siempre prepara 24 líneas de GRAPHICS 0. En este aspecto es muy estricto y, aunque se modifique el DL, no por ello cambiará sus hábitos de trabajo.

Además existen otros problemas. En GRAPHICS 2 sólo se puede acceder a la mitad del juego de caracteres y la modificación efectuada ahora sólo admite una línea física de GRAPHICS 0, es decir que no acepta una línea lógica completa. Sin embargo, es posible convertir toda la pantalla de GRAPHICS 0 en GRAPHICS 2, permitiendo así escribir programas en GRAPHICS 2. Ello puede ser de ayuda para personas cortas de vista. Si quiere limitarse a destacar una línea física, es aconsejable disponerla en el borde superior de la pantalla.

MEMORIA DE PANTALLA

Con el nombre de screen memory (SM, memoria de pantalla) se designa aquella zona ocupada por los datos que definen la configuración de la pantalla. La instrucción LMS del DL espera encontrar datos gráficos. La manera de transformar estos datos en informaciones gráficas está determinada por el comando de la línea de modo.

Existen dos modos operativos en principio diferentes, el modo carácter y el modo bit-map.

En el modo carácter, cada byte de datos leído en la SM es transformado en un carácter. Un carácter es una disposición determinada de puntos de imagen. La relación entre los caracteres, los valores que los representan y los esquemas de puntos que los representan en pantalla, está definida en el juego de caracteres (vea apéndice juego de caracteres). Los caracteres del juego están ordenados por valores de 0 a 127. Sin embargo, estos valores difieren del código ATASCII. Se habla del código interno.

En el modo carácter el valor de la memoria de pantalla es interpretado como código interno de un carácter, se lee el carácter del juego y se emite en pantalla el bit tipo que lo representa.

El comando BASIC LOCATE x,y,n , busca en la memoria de pantalla la celda correspondiente a la posición de pantalla x,y , lee el valor allí colocado (código interno) y lo transforma en ATASCII. LOCATE localiza pues en el modo carácter el valor ATASCII del carácter que se encuentra en cierta posición de la pantalla.

Por cierto, el comando LOCATE sólo trabaja si previamente se ha efectuado un comando GRAPHICS. Esto significa que incluso GRAPHICS 0 debe programarse expresamente, cosa innecesaria habitualmente.

También GRAPHICS 1 y 2 son modos carácter. La diferencia estriba en que sólo se leen los bits 0 a 5 para encontrar el carácter en el juego de caracteres. Por esta razón no se puede representar más que medio juego de caracteres, de código interno 0 a 63. Los bits 6 y 7 son interpretados como información de color. Con dos bits pueden representarse cuatro valores diferentes (0 a 3). Por ello, los caracteres pueden tomar cuatro valores diferentes en estos dos modos operativos. Un quinto color es asignado al fondo.

Los modos carácter 12 y 13 también representan los caracteres de varios colores, permitiendo sin embargo llamar al juego de caracteres completo, puesto que la información de color se consigue por otra vía. En un carácter estándar, cada bit corresponde a un punto de imagen, por lo que puede aparecer en dos colores, a saber el color del carácter o el del fondo. En estos dos modos gráficos cada dos bits se emiten en un solo punto de imagen, pudiendo acceder de esta forma a cuatro valores de color diferentes (registros de color). Al representar los caracteres del juego estándar en este modo devienen prácticamente ilegibles, porque la forma gráfica es traducida sólo en parte, mientras que se añaden interpretaciones de color sin sentido. (Más detalles en el apéndice juego de caracteres)

En modo bit-map, se emiten pixels en la pantalla de tv cuyo tamaño depende de la resolución del modo gráfico escogido.

Si cada bit representa un punto gráfico, éste sólo podrá tomar dos colores, 0 ó 1. El COLOR contenido en 0 y 1 se define con ayuda de POKE o SETCOLOR en el registro de color (vea tabla) al que se ha accedido mediante COLOR. En un byte de esta SM caben ocho de estos bits gráficos. Según la situación del bit en el interior del byte de memoria o la del pixel en la pantalla, el bit recibe un valor posicional de la manera habitual:

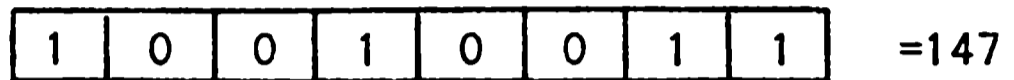
.

DOS COLORES

Pixel en la pantalla



Bit tipo



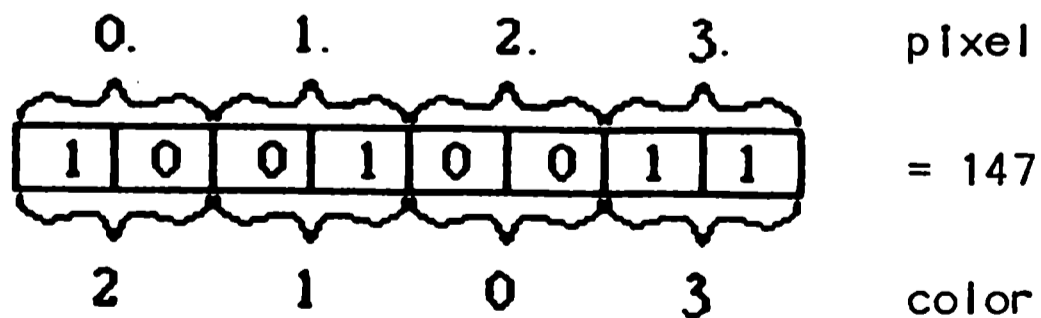
Valor posicional: 128 64 32 16 8 4 2 1

El valor 147 de una celda de la SM origina en la pantalla la secuencia de pixels antes mostrada.

Los modos gráficos 0, 4, 6, 8 y 14 trabajan en dos colores.

Si hay que distinguir cuatro colores, cada pixel requiere dos bits de información:

CUATRO COLORES

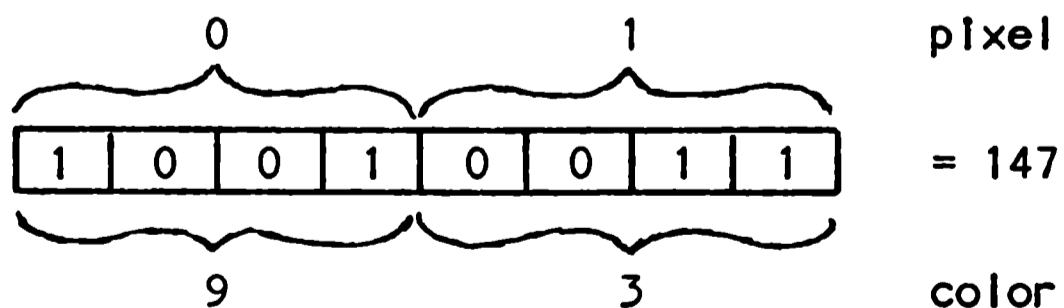


De esta forma, el byte de datos 147 origina una secuencia de cuatro pixels en los colores 2, 1, 0, 3. En la memoria de pantalla se coloca pues el valor de COLOR del pixel en forma binaria. A través del valor de COLOR se determinarán los correspondientes registros de color. Puesto que el comando PEEK halla sólo el valor decimal de un byte, los bits de varios pixels forman un valor decimal común.

Los modos gráficos 3, 5, 7 y 15 trabajan con dos bits por pixel.

En los modos GITA 9, 10 y 11 se usan cuatro bits por cada pixel. Con cuatro bits se pueden distinguir cuatro colores. Sin embargo, el ATARI sólo dispone de nueve registros de color, por lo que es imposible definir dieciseis colores. Por esta razón COLOR 0 a 15 en GRAPHICS 9 sólo hacen referencia a diversas gradaciones de claridad de un color determinado; en GRAPHICS 11, en cambio, se pueden utilizar los dieciseis colores estándar pero todos con la misma claridad; y en GRAPHICS 10 se permiten definir libremente los valores de color, pero al disponer únicamente de nueve registros de color sólo pueden efectuarse nueve comandos COLOR:

16 COLORES



En modo GITA, el valor 147 origina dos pixel con los valores de COLOR 9 y 3.

La propia memoria de pantalla está dispuesta linealmente. El primer byte de SM determina el contenido de la esquina superior izquierda de la ventana gráfica. Le siguen los datos correspondientes a la línea superior hasta el extremo derecho, después los datos de la líneas segunda y siguientes hasta llegar al último valor, que define la configuración de la esquina inferior derecha.

El número de bytes ocupados por una línea depende del tamaño del pixel y de la cantidad de colores posibles, es decir de los bits por pixel.

En GRAPHICS 3 una línea se compone de 40 pixel; ya que se admiten cuatro colores diferentes, cada pixel ocupa dos bits. Una línea del modo gráfico 3 ocupa 80 bits = 10 bytes. Para acceder a un punto determinado de la pantalla hay que averiguar qué byte contiene los datos correspondientes a este punto. Si el punto tiene las coordenadas X,Y sus datos se encuentran en el:

$$\text{byte} = \text{SAVMSC} + Y * \text{rpl} * \text{INT}(X / \text{ppb})$$

donde rpl define la RAM por línea o los bytes por línea y ppb indica los pixel por byte. SAVMSC es el vector que apunta hacia la dirección inicial de la memoria de pantalla. El valor de este puntero apunta hacia el primer byte de la SM.

Una vez hallado el byte correcto, habrá que determinar además el lugar, dentro del byte, donde se encuentran los datos del punto gráfico en cuestión.

En GRAPHICS 3 se pretende colocar un pixel de COLOR 2 en la posición 5,2. Dicha posición provoca un offset de SAVMSC de 21 bytes. En el byte 21 debajo de SAVMSC se hallan los pixel con las coordenadas de pantalla siguientes:

Bit	7	6	5	4	3	2	1	0
Valor decimal	128	64	32	16	8	4	2	1
POSITION	-4, 2-	-5, 2-	-6, 2-	-7, 2-				
COLOR	0 0	1 0	0 0	0 0				

En la posición 5,2 se ha depositado el valor de COLOR 2 (en binario 10). Si los tres puntos restantes permanecen negros (fondo, COLOR 0), el byte 21 tomará el valor 32. El comando BASIC:

```
COLOR 2:PLOT 5,2
```

puede sustituirse por el comando:

```
POKE SM+21,32
```

calculando previamente SM como dirección inicial de la pantalla a partir del vector SAVMSC:

```
SM = PEEK(88)+PEEK(89)*256
```

Sin embargo, si hay que proyectar puntos gráficos aislados en la pantalla, el comando PLOT es mucho más fácil de manejar. Pero también el POKE hacia la SM tiene algunas ventajas.

POKE es independiente de las posiciones del cursor que pueden ser consideradas ilegales por el OS después de modificar el DL. Ello significa que POKE sirve para escribir en zonas de la memoria inaccesibles para PLOT y DRAWTO.

Otra aplicación interesante consiste en almacenar contenidos de pantalla, PEEKeando byte por byte de la memoria de pantalla y grabándolos (PUT) en el diskette a través de un canal de datos. En sentido Inverso, se podrán GETear estos datos byte por byte del fichero del diskette y POKEarlos en la SM.

El siguiente programa ejemplo muestra una aplicación de direccionamiento directo de la memoria de pantalla:

```
0 REM SMPOKE.BEC
1 REM *****
2 REM *
3 REM * LETRERO CON MOVIMIENTO *
4 REM *
5 REM *****
10 S=40040:PRINT CHR$(125)
20 FOR P=0 TO 30:READ B:POKE S+P,B:NEXT P
30 DATA
0,10,0,10,0,36,105,101,115,116,0,101,105,110,0,44,97,117,102,
115,99,104,114,105,102
40 DATA 116,13,52,101,115,116,0,10,0,10,0,10
50 J=PEEK(S):K=PEEK(S+1):L=PEEK(S+2):M=PEEK(S+3)
60 FOR I=4 TO 39
70 POKE S+I-4,PEEK(S+I)
80 NEXT I
90 POKE S+36,J:POKE S+37,K:POKE S+38,L:POKE S+39,M
100 GOTO 50
```

10: S es una dirección de SM.

20: En el bucle FOR-NEXT, las celdas de memoria SM leen los bytes de datos correspondientes al texto deseado para una línea de pantalla (0 a 39).

30 y 40: Los valores de carácter deben corresponder al código interno.

50: El contenido de las primeras cuatro direcciones es asignado a variables.

60 a 80: A continuación se trasladan los contenidos de las celdas restantes de la línea de pantalla en cuatro direcciones hacia arriba, que corresponde en la pantalla a cuatro columnas hacia la izquierda, y

90: finalmente los cuatro valores de carácter reservados son POKEados en las cuatro últimas posiciones de la línea.

Mediante este proceso los caracteres de la línea de pantalla se trasladan cuatro columnas hacia la izquierda con cada ejecución del programa. También es posible, naturalmente, trasladarlos sólo en un carácter cada vez, pero el BASIC es tan lento que el texto se desplazaría a cámara lenta.

Sin embargo, el direccionamiento directo de la pantalla para la ventana de texto tiene todavía más importancia, puesto que en ella no se activa el comando POSITION. Utilizando un POKE SM se pueden colocar caracteres aleatorios y seguros y emitir, así, valores continuamente cambiantes en un mismo lugar, como puede ser por ejemplo un contador:

```
0 REM RWPOKE.BEC
1 REM *****
2 REM * *
3 REM * POKES A LA VENTANA DE TEXTO*
4 REM * *
5 REM *****
10 DIM Z$(5):P=40897:PRINT CHR$(125):N=25
20 FOR J=0 TO 4:POKE P+J,16:NEXT J
30 Z=Z+1
40 Z$=STR$(Z)
50 L=LEN(Z$)
60 FOR J=1 TO L
70 POKE P+4-L+J,VAL(Z$(J,J))+16
80 NEXT J
90 IF PEEK(P)=N THEN IF PEEK(P+1)=N THEN IF PEEK(P+2)=N THEN
IF PEEK(P+3)=N THEN IF PEEK(P+4)=N THEN GOSUB 200
100 REM IF Z=99999 THEN GOSUB 200
110 GOTO 30
200 RESTORE 300
210 FOR J=0 TO 8
220 READ D
230 POKE P-10+J,D
240 POKE P+6+J,D
250 NEXT J
260 Z=0:FOR J=0 TO 4
270 POKE P+J,16
280 NEXT J
290 RETURN
300 DATA 47,54,37,50,38,44,47,55,1
```

10: La secuencia de cifras correspondiente al estado del contador es asignada a Z\$; en todos los DL estándar la esquina superior izquierda de la ventana de texto se encuentra en la dirección 40800.

20: Se escriben 0 en las cinco primeras posiciones del contador; su código interno = 16.

30: Z cuenta.

40: Para POKEar el estado del contador en la pantalla hay que descomponer Z en cifras separadas. El primer paso consiste en convertir la variable numérica Z en un string Z\$, porque es fácil leer en un string cada uno de los elementos.

50: Para conseguir que cada cifra llegue a parar en la posición correcta del número emitido, hay que determinar en primer lugar cuántas cifras tiene el número en cuestión, es decir cuántos elementos contiene Z\$.

60 a 80: El bucle FOR-NEXT lee en los strings de cifras cada una de las mismas (Z\$(J,J)), las convierte en valor numérico (VAL) y suma 16 a cada cifra para transformarla en su código interno de carácter. El código interno de 1 es 17, el de 2 es 18, etc. El valor de carácter así determinado es POKEado en la posición P+4-L+J. De esta forma se coloca cada cifra en su lugar correspondiente.

90: El ATARI admite también una instrucción como ésta. Sirve para determinar si todas las cifras tienen valor 9 (código interno = 25).

100: También puede conseguirse más fácilmente.

110: Cierra el bucle con un salto hacia la línea 30.

200 a 300: Cuando el contador haya llegado hasta 99999 en este lugar se ... no, no lo descubrimos - usted mismo puede averiguar lo que ocurre. Sin embargo, si hace llegar el contador hasta el final tendrá que armarse de un poco de paciencia.

Al cambiar de 99999 a 00000 podrá observar que el contador va mucho más deprisa en los números pequeños. Ello se debe a la menor longitud del string de cifras, con lo que se precisan menos pasos de programa. El BASIC es suficientemente lento para permitir que se aprecie esta diferencia con la vista.

La ventana de texto presenta otra particularidad, y es que dispone de una memoria de pantalla propia e independiente. El vector TXTMSC (660,661 = \$294,\$295) apunta hacia la dirección inicial, que en todos los DL estándar es la 40800.

Independientemente de reclamar un modo gráfico con o sin (+16) ventana de texto, siempre se reserva memoria para la ventana gráfica capaz de almacenar los datos necesarios para todo el volumen de la pantalla. Detrás de ella se encuentra la memoria de 160 bytes reservada para la ventana de texto. Esta independencia también es la razón por la que aquí no se activan los PLOTS y POSITIONS.

Por otro lado, esta separación abre la posibilidad de escribir en la memoria de la ventana de texto sin hallarse ésta conectada, así como de visualizar y ocultar la ventana de texto en la ventana gráfica sin que, por ello, se pierdan datos de una o de otro. Sin embargo, este proceso sólo es posible si al realizar el comando GRAPHICS se prevé que el contenido de la memoria no sea borrado como es habitual. Ello se consigue activando el bit 5 del número de identificación de GRAPHICS:

Bit 7 no utilizado

Bit 6 no utilizado
Bit 5 (=64) no borrar memoria de pantalla
Bit 4 (=16) sin ventana de texto
Bit 3
 a (=15 a 0) modo gráfico 0 a 15 con ventana de texto
Bit 0

El siguiente ejemplo le muestra los pasos de programa a realizar para visualizar y ocultar la ventana de texto:

```

0 REM TWEINAUS-BEC
1 REM *****
2 REM * *
3 REM * VISUALIZAR/OCULTAR *
4 REM * VENTANA DE TEXTO *
5 REM * *
6 REM *****
10 GOSUB 300
20 Z=Z+1:Z$=STR$(Z):L=LEN(Z$)
30 FOR J=1 TO L:POKE P+6-6+J,VAL(Z$(J,J))+16:NEXT J
40 IF PEEK(764)=225 THEN 20
100 OPEN #1,4,0,"K:":GET #1,T:CLOSE #1
110 YA=Y:Y=Y-((T=28) AND (Y>0))
120 Y=Y+((T=29) AND (Y<23))
130 XA=X:X=X-((T=30) AND (X>0))
140 X=X+((T=31) AND (X<39))
150 IF T=69 THEN GRAPHICS 3+32:POKE 708,36:POKE 709,38:POKE
710,208:POKE 712,208
160 IF T=65 THEN GRAPHICS 3+16+32:POKE 708,50:POKE
709,52:POKE 710,208:POKE 712,208
200 COLOR 0:PLOT 0,YA:DRAWTO 39,YA:PLOT XA,0:DRAWTO X,23:GOTO
20
210 COLOR 1:PLOT 0,Y:DRAWTO 39,Y:COLOR 2:PLOT X,0:DRAWTO
X,23:GOTO 20
300 DIM Z$(7):P=40896
310 GRAPHICS 10:POKE 708,50:POKE 709,52:POKE 710,208:POKE
712,208
320 FOR J=0 TO 6:POKE P+J,16:NEXT J
330 COLOR 2:PLOT 0,0:DRAWTO 0,23
340 COLOR 1:PLOT 0,0:DRAWTO 39,0
350 RETURN

```

10: No es siempre aconsejable disponer el programa en orden cronológico, es decir comenzar con la introducción en la primera línea. ¿Por qué? Continúe leyendo:

20 y 30: Contienen las mismas instrucciones que el contador antes comentado, ampliado únicamente a siete cifras y ocupando menos líneas lógicas. La ejecución de un programa BASIC es tanto más rápida cuanto menos líneas lógicas contenga, en la medida en que existan instrucciones de salto en el programa, puesto que cada salto da lugar a que el

BASIC revise todos los números de línea, de arriba a abajo hasta encontrar la línea mencionada. Por ello también es aconsejable disponer las líneas más solicitadas por los saltos al principio del programa, y las menos activas o las de ejecución única al final del mismo.

40: CH (764 = \$2FC) contiene el código de teclado de la última tecla pulsada. Su valor es 255 mientras no se pulse ninguna tecla. Esta línea se encarga de que el programa proceda su ejecución sin ningún problema, ya que

100: el comando GET interrumpe el programa y espera una entrada efectuada por el usuario. Sin embargo, durante este tiempo el contador también estaría parado.

110 a 140: En la ventana gráfica se encuentra un retículo. Podrá moverlo en todas direcciones con ayuda de las teclas de control del cursor. XA e YA mantienen los valores antiguos que permiten borrar más tarde la posición inicial del retículo. Las cuatro operaciones booleanas determinan si se ha pulsado la tecla en cuestión (T) y si no se han alcanzado los valores límite de X o Y, actualizando además correspondientemente los valores de X y de Y.

150: Si la tecla pulsada es la "E" se activa GRAPHICS 3 (o sea con ventana de texto). El +32 se encarga al mismo tiempo de que la pantalla no sea borrada. En una palabra, se visualiza la ventana de texto. También puede escribir GRAPHICS 35 de entrada en lugar de 3+32.

Sin embargo, cada comando GRAPHICS renueva los valores default a los registros de color. Por esta razón hay que renovar aquí los colores definidos. Por cierto, utilizando el mismo valor de color para el fondo (712) y COLOR 3 (710) se consigue que las ventanas gráfica y de texto formen una unidad óptica, aumentando así el efecto de visualización.

Sin duda sería todavía más impresionante estrechar la ventana de texto. Para conseguir este efecto bastaría una única línea. Puesto que para ello habría que modificar el DL, la línea a visualizar podría situarse además en cualquier posición de la pantalla.

160: Si se pulsa la tecla "A", la ventana de texto vuelve a desaparecer gracias al número de identificación de GRAPHICS 3+16+32 (=51). Puesto que al visualizar y ocultar hay que volver a escribir en los registros de color, el cursor adquiere un color ligeramente diferente cuando se visualiza el contador.

Es importante que, incluso con la ventana de texto visualizada, se pueda mover el centro del retículo hasta llegar al borde inferior de la pantalla, es decir hasta el lugar donde se extiende ahora la ventana de texto, sin que se produzca un mensaje de error por posición ilegal del cursor. Sin embargo, ello sólo es posible si al principio se activó el modo gráfico sin ventana de texto. Aunque se visualice después la ventana de texto, se podrá escribir igualmente con PLOT, etc. en toda la ventana gráfica incluyendo la zona ocupada por la ventana de texto, aunque no sea visible temporalmente.

200: Escribe con el color de fondo encima del retículo antiguo, es decir que lo borra, y la línea

210: dibuja (PLOT) el retículo actual.

300 a 350: Las instrucciones utilizadas únicamente al iniciar el programa están mejor guardadas al final del mismo. Aquí se definen gráficos y valores de color, se colocan 0 en las siete posiciones del contador y se dibujan (PLOT) los dos ejes del retículo.

La siguiente tabla sirve de esquema de los diversos modos gráficos:

A debe salir Inverso

TABLA

GRAPHICS	ANTIC Comando	Columnas pixel por línea	Líneas de modo	Bytes por línea	Líneas TV por línea de modo	Bits por pixel	Registros de color			
							708	709	710	711
0	2	40	20/24	40	8		C.1*	C.0*	711	712**
-	3	40	16	40	10		C.1	C.0		R
12	4	40	20/24	40	8	A/A	A/A	A	A	H
13	5	40	10/12	40	16	A/A	A/A	A	A	H
1	6	20	20/24	20	8	A	a	A	a	H
2	7	20	10/12	20	16	A	a	A	a	H
3	8	40	20/24	10	8	C.1	C.2	C.3		H
4	9	80	40/48	10	4	C.1				H
5	10	80	40/48	20	4	C.1	C.2	C.3		H
6	11	160	80/96	20	2	C.1				H
14	12	160	160/192	20	1	C.1				H
7	13	160	80/96	40	1	C.1	C.2	C.3		H
15	14	160	160/192	40	1	C.1	C.2	C.3		H
8	15	320	160/192	40	1		C.1	C.0		R
9		80	192	40	1	16 gradaciones de claridad				
10		80	192	40	1	9 matices de color				
11		80	192	40	1	16 colores igual claridad				

- * En todos los GRAPHICS con ventana de texto el número 710 determina los colores de caracteres y de fondo y el 709 la claridad de caracteres de la ventana de texto.
C = COLOR, R = BORDE, H = FONDO.

- ** Con COLOR 0 se llama al valor de color del registro 712 en todos los GRAPHICS.

JUEGO DE CARACTERES

Para ser capaz de emitir información escrita en la pantalla, el ordenador debe conocer en primer lugar el aspecto de los caracteres utilizados para la comunicación humana, ya que el mismo ordenador se limita a usar números.

Existen tres sistemas diferentes para asignar valores numéricos al surtido de caracteres, el character set; todos ellos tienen validez en diferentes niveles del sistema.

Por un lado existe el código de teclado que asigna un valor numérico de 0 a 63 a cada tecla. Ya desde los tiempos de la máquina de escribir, la segunda mitad del juego de caracteres es accesible con ayuda de una tecla de función, que ha adquirido el nombre de SHIFT, pero con la diferencia de que en el estado normal del teclado, el ATARI emite letras mayúsculas. Utilizando la tecla CAPS se puede bloquear el modo máquina de escribir, y el teclado emitirá inúsculas y mayúsculas (con SHIFT).

Al ordenador se añadió entonces otra tecla de función para permitir que cada tecla creara tres caracteres distintos. Su designación habitual es CONTROL o CTRL.

Una tercera tecla común a todos los ordenadores afecta al aspecto de los caracteres. Es la tecla INVERS. Al pulsarla se invierten todos ellos, es decir se muestran en representación negativa en la pantalla.

Desde que se introdujo el uso del teletipo, las empresas, normalmente propensas a aprovechar cualquier oportunidad para crear sus estándares individuales, fueron obligadas a adaptarse a una sola norma para garantizar también la transferencia de información entre distintas marcas. Este estándar recibe el nombre de código ASCII, que asigna un valor numérico de 0 a 127 a cada carácter y a cada función del teletipo, tales como avance de línea, tabulador, etc.

Actualmente el código ASCII también constituye la base para impresoras y computadoras. Más tarde se añadieron a esta norma americana caracteres internacionales y con ello también españoles (ñ, ¿, ¡, acentos, etc.). Sin embargo, éstos ya no cupieron entre los primeros 128 caracteres, es decir en el juego de caracteres propiamente dicho.

Los primeros 32 caracteres del código ASCII sirven para el control funcional de la cabeza de impresión. El ordenador no precisa de éstos caracteres de control puesto que emite a través del monitor. Los fabricantes de ATARI han aprovechado este espacio libre para albergar los caracteres adicionales que un teletipo formal está lejos de utilizar, siendo sin embargo útiles para un dispositivo de ocio. Se trata de los llamados pseudo- o semi-caracteres o caracteres de gráficos de bloque. Para diferenciar este código de la norma se hace referencia al código ATASCII, que en gran parte es idéntico a la norma ASCII.

Sin embargo, para dificultar la labor del programador aficionado, existe un tercer sistema designado con el nombre de código interno. El código interno determina el orden en que se almacenan los datos de los 128 caracteres en la memoria ROM.

Este código interno se distingue del código ATASCII por el hecho de que la suma de los caracteres está intercambiada en tres grandes bloques. La relación entre ambos códigos se presenta de la manera siguiente:

Conversión de valores ATASCII en código interno

0 - 31	y	128 - 159	ATASCII + 64
32 - 95	y	160 - 223	ATASCII - 32
96 - 127	y	224 - 255	Idéntico

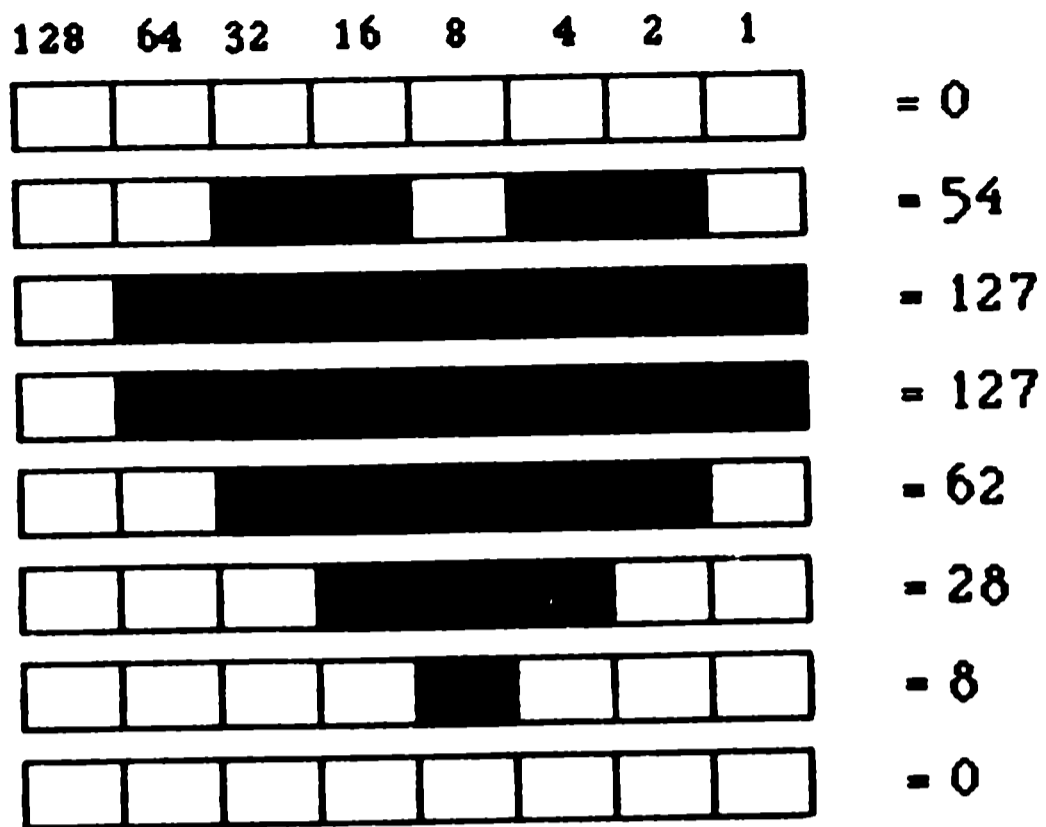
Conversión de código interno a valores ATASCII

0 - 63	y	128 - 191	código interno + 32
64 - 95	y	192 - 223	código interno - 64
96 - 127	y	224 - 255	Idéntico

El carácter cero del código interno corresponde al espacio (ATASCII 32). Es decir que el juego de caracteres comienza con los datos que hacen referencia al espacio.

Cada carácter es representado en una matriz de ocho por ocho puntos. Cada uno de los puntos reluce o no; este dato se almacena activando o desactivando un bit. Ocho bits dispuestos uno tras otro se reúnen en un byte de datos. Cada carácter tiene pues una altura de ocho bits:

El carácter ATASCII 0



El ordenador memorizará este corazón en forma de secuencia de ocho bits cuyos valores decimales son 0, 54, 127, 127, 62, 28, 0. Tiene valor ATASCII 0 y de código Interno 64. Esto significa que se encuentra en la posición número 65 de la memoria del juego de caracteres. Sin embargo, puesto que cada uno de los 64 caracteres precedentes también ocupa ocho bits, el primer byte del corazón se encuentra en la 64*8 celda de memoria detrás de la dirección inicial del juego de caracteres. El código Interno determina pues el offset de los datos de un carácter en la memoria de caracteres. El byte HI de la dirección inicial del juego se encuentra en CHBAS (756 = \$2F4). Para hallar los datos de un carácter determinado se calcula:

$$\text{CHBAS} * 256 + \text{código Interno del carácter} * 8$$

Los datos del carácter en cuestión se encontrarán en ésta y en las siete posiciones siguientes.

Una vez sabido esto, la creación de un juego de caracteres individual ya sólo es cuestión de trabajo activo. Para ello "simplmente" hay que calcular el byte de datos a partir de la muestra de ocho por ocho puntos para cada uno de los 128 caracteres, y almacenar estos $8*128 = 1024$ ó 1 kbyte en la memoria. Si después se dirige el puntero CHBAS hacia la dirección inicial de este juego de caracteres alternativo, cada PRINT y cada listado aparecerá con un nuevo aspecto en la pantalla.

El siguiente programa le muestra un ejemplo de ello, el cual contiene un juego de caracteres en cursiva en forma de líneas DATA. Un carácter en cursiva se distingue porque tiene su eje vertical inclinado hacia la derecha, tal como suele ser característico en la letra manuscrita. Los americanos la llaman "italic", sabrá Zeus por qué:

```
0 REM ITALIC.DAT
1 REM *****
2 REM * *
3 REM * SET DE CARACTERES (DATA) *
4 REM * CURSIVOS *
5 REM * *
6 REM *****
10 POKE 106,PEEK(106)-8
20 GRAPHICS 0
30 A=PEEK(106)+8:C=A*256
40 FOR J=0 TO 1023:READ B:POKE C+J,B:NEXT J
50 PRINT CHR$(125):POKE 752,1
100 POKE 756,A
110 POSITION 2,7
120 PRINT " ASI SE REPRESENTA LA LETRA CURSIVA."
130 FOR W=0 TO 400:NEXT W
140 POSITION 2,7
150 PRINT "ASI SE REPRESENTA LA LETRA NORMAL."
```



```

· 160 POKE 756,224
170 FOR W=0 TO 400:NEXT W
180 GOTO 100
30000 DATA
0,0,0,0,0,0,0,0,0,12,12,24,24,0,48,0,0,51,51,102,0,0,0,0
30001 DATA
0,54,127,54,108,254,108,0,12,62,96,56,12,248,48,0,0,108,104,2
4,16,54,38,0
30002 DATA
24,52,24,40,74,68,54,0,0,24,24,48,0,0,0,0,14,28,24,24,28,14
,0
30003 DATA
0,112,56,24,24,56,112,0,0,102,60,255,60,102,0,0,0,8,24,126,24
,16,0,0
30004 DATA
0,0,0,0,0,24,24,48,0,0,0,126,0,0,0,0,0,0,0,24,24,0
30005 DATA
0,6,12,24,48,96,64,0,0,28,50,110,118,76,56,0,0,6,28,12,24,24,
48,48
30006 DATA
0,30,50,12,24,48,124,0,0,31,6,8,4,4,40,48,0,6,12,28,40,126,24
,48
30007 DATA
0,62,48,56,4,4,76,48,12,16,32,120,100,100,56,0,0,60,4,12,24,4
8,48,48
30008 DATA
12,18,50,56,76,76,56,0,0,28,54,34,22,12,24,48,0,0,24,24,0,48,
48,0
30009 DATA
0,0,24,24,0,48,48,96,6,12,24,48,24,12,6,0,0,0,126,0,0,126,0,0
30010 DATA
96,48,24,12,24,48,96,0,0,28,34,12,24,16,0,32,0,60,54,110,108,
64,124,0
30011 DATA
0,15,27,50,102,252,204,0,0,62,50,124,102,198,252,0,0,30,50,32
,96,100,56,0
30012 DATA
0,60,54,102,102,204,248,0,0,62,48,120,96,192,248,0,0,62,48,12
0,96,192,192,0
30013 DATA
0,30,48,96,110,204,120,0,0,51,51,126,102,204,204,0,0,30,12,24
,24,48,248,0
30014 DATA
0,30,38,12,12,24,152,112,0,51,54,104,120,216,204,0,0,24,24,48
,48,96,124,0
30015 DATA
0,33,51,127,107,198,198,0,0,51,59,106,110,204,196,0,0,30,51,1
02,102,204,120,0
30016 DATA
0,30,19,54,60,96,96,0,0,30,51,102,102,216,108,6,0,62,51,102,1
24,216,204,0
30017 DATA
0,30,48,60,6,140,120,0,0,63,76,24,24,48,48,0,0,51,99,102,198,
204,120,0
30018 DATA
0,51,35,102,100,104,48,0,0,33,99,67,203,222,228,0,0,34,22,28,
24,52,100,0
30019 DATA
0,17,35,22,28,24,48,96,0,62,6,24,48,192,248,0,0,30,24,48,48,9
6,120,0
30020 DATA
0,96,96,48,24,12,12,0,0,30,6,12,12,24,120,0,0,8,28,54,99,0,0,
0
30021 DATA
0,0,0,0,0,0,255,0,0,54,127,127,62,28,8,0,24,24,24,31,31,24,24
,24
30022 DATA
3,3,3,3,3,3,3,3,24,24,24,248,248,0,0,0,24,24,24,248,248,24,24
,24
30023 DATA
0,0,0,248,248,24,24,24,3,7,14,28,56,112,224,192,192,224,112,5
6,28,14,7,3
30024 DATA
1,3,7,15,31,63,127,255,0,0,0,0,15,15,15,15,128,192,224,240,24
8,252,254,255

```

30025 DATA
 15,15,15,15,0,0,0,0,240,240,240,240,0,0,0,0,255,255,0,0,0,0,0
 ,0
 30026 DATA
 0,0,0,0,0,0,255,255,0,0,0,0,240,240,240,240,0,28,28,119,119,8
 ,28,0
 30027 DATA
 0,0,0,31,31,24,24,24,0,0,0,255,255,0,0,0,24,24,24,255,255,24,
 24,24
 30028 DATA
 0,0,60,126,126,126,60,0,0,0,0,255,255,255,255,192,192,192,1
 92,192,192,192,192
 30029 DATA
 0,0,0,255,255,24,24,24,24,24,24,255,255,0,0,0,240,240,240,240
 ,240,240,240,240
 30030 DATA
 24,24,24,31,31,0,0,0,120,96,120,96,126,24,30,0,0,24,60,126,24
 ,24,24,0
 30031 DATA
 0,24,24,24,126,60,24,0,0,24,48,126,48,24,0,0,0,24,12,126,12,2
 4,0,0
 30032 DATA
 0,24,60,126,126,60,24,0,0,0,28,6,62,204,122,0,0,48,48,124,102
 ,204,248,0
 30033 DATA
 0,0,60,96,96,192,120,0,0,3,3,62,102,204,124,0,0,0,28,102,124,
 192,120,0
 30034 DATA
 0,14,24,60,24,48,48,96,0,0,31,54,102,60,12,120,0,48,48,124,10
 2,198,204,24
 30035 DATA
 0,12,0,56,24,48,120,0,0,6,0,12,12,24,24,240,0,48,32,108,120,2
 08,204,0
 30036 DATA
 0,28,12,24,24,48,120,0,0,0,51,127,122,214,132,0,0,62,99,103
 ,198,204,0
 30037 DATA
 0,0,30,102,102,204,120,0,0,0,62,102,102,248,192,192,0,0,31,10
 2,102,60,12,12
 30038 DATA
 0,0,26,48,48,96,96,0,0,0,31,96,60,12,248,0,0,12,63,24,24,48,4
 8,0
 30039 DATA
 0,0,51,102,102,204,124,0,0,0,51,102,102,120,48,0,0,0,35,107,8
 7,252,216,0
 30040 DATA
 0,0,51,60,24,124,196,6,0,0,51,98,102,60,24,240,0,0,63,12,24,9
 6,252,0
 30041 DATA
 0,24,60,126,126,24,60,0,24,24,24,24,24,24,24,0,126,120,124
 ,110,102,6,0
 30042 DATA
 3,24,56,120,56,24,8,0,255,255,255,255,255,255,255,255

10: Es especialmente aconsejable almacenar el valioso juego de caracteres en una zona protegida de la memoria. Cuando haya tecleado este programa, o cuando haya programado incluso un juego de caracteres propio, se dará cuenta de lo valioso que es un juego de este tipo.

El vector de RAMTOP (106 = \$6A) es trasladado ocho páginas hacia abajo. A decir verdad, sólo se precisan cuatro páginas (= 1 kbyte), pero es más seguro reservar un zona de buffer al acercarse a la memoria de la ventana de texto, para

evitar que el scroll pueda ocultar los datos del juego de caracteres (sólo ≠).

20: El comando GRAPHICS da lugar a que la memoria de pantalla y el display list se subordinen al nuevo valor de RAMTOP. Por cierto, el nuevo juego de caracteres también funciona en GRAPHICS 1 ó 2 o en cualquier modo gráfico con ventana de texto. Simplemente tenga en cuenta que se restablece el valor default de CHBAS con cada comando GRAPHICS, por lo que deberá volver a POKEar la dirección inicial del juego de caracteres alternativo en la dirección 765.

30: Se calculan los dos valores necesarios, o sea la página o byte HI y la dirección inicial completa del nuevo juego de caracteres.

40: En el bucle FOR-NEXT la zona de memoria reservada lee ahora los 1024 bytes en las líneas DATA. Este proceso requiere cierto tiempo. Pero una vez transcurrido ya puede comenzar.

50: Se limpia la pantalla y se oculta el cursor.

100: Aquí se dirige el puntero hacia nuestro nuevo juego de caracteres y

120: la pluma del abuelo escribe en el monitor:

150: A continuación escribimos otro texto y

160: dirigimos el puntero de nuevo hacia el juego de caracteres estándar protegido en la ROM. En un instante todo vuelve a la normalidad.

30000 a 30042: Este es byte por byte el juego de caracteres *Italic*.

Y así aparecen los caracteres nuevos y los estándar del ATARI en la pantalla:







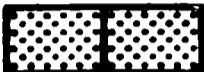

























* Dibujo de los juegos de caracteres *

Los modos carácter 12 y 13, que los XL también pueden llamar a través de GRAPHICS, funcionan con el mismo principio. La diferencia estriba en que cada dos bits crean un pixel del carácter. Ello permite distinguir un total de cuatro valores de COLOR, pero un byte sólo almacenará las informaciones correspondientes a cuatro pixels. Para conseguir la misma anchura de display los pixels tienen un ancho de dos puntos de imagen.

Si los valores del juego de caracteres estándar se emiten de esta forma, aparecerán disposiciones poco razonables de puntos de color. Estos dos modos GRAPHICS sólo serán realmente efectivos con un nuevo juego de caracteres.

El corazón ATARI por ejemplo tiene el aspecto siguiente en GRAPHICS 12 o 13:

Caracteres en GRAPHICS 12 y 13

PIXELS				Valores COLOR	valores decimales
				- 0 0 0 0	- 0
				- 0 3 1 2	= 54
				- 1 3 3 3	- 127
				- 1 3 3 3	- 127
				- 0 3 3 2	- 62
				- 0 1 3 0	- 28
				- 0 0 2 0	- 8
				- 0 0 0 0	- 0

Los valores de COLOR hacen referencia a los registros de color 708 (COLOR 1 = 01), 709 (2 = 10), 710 (3 = 11) y fondo 712 (0 = 00) de los caracteres normales. Los Inversos obtienen su valor para COLOR 3 del registro de color 711.

Para finalizar le presentamos un pequeño programa que muestra los efectos que pueden conseguirse utilizando caracteres de varios colores:

```

0 REM COLCHR.DAT
1 REM *****
2 REM *
3 REM * CARACTERES DE COLOR
4 REM *
5 REM *****
10 GOSUB 100
20 X=INT(RND(0)*39)
30 Y=INT(RND(0)*21)
40 Z=INT(RND(0)*11)
50 J=Y*40+X+1:Z=Z+1
60 P$(J,J)=C$(Z,Z)
70 POSITION 0,0:PRINT# 6;P$:GOTO 20
100 DIM P$(880),C$(11):C$="#$%&'()*+,-"
110 P$="#":P$(880)=P$:P$(2)=P$
120 POKE 106,PEEK(106)-4
130 GRAPHICS 28
140 A=PEEK(106)+4:C=A*256
150 FOR J=0 TO 110:READ B:POKE C+J,B:NEXT J
200 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
210 DATA
66.66,24,24,36,36,129,129,78,78,116,116,39,39,177,177,112,112
,27,27,228,228,141,141
220 DATA
74,74,16,16,4,4,161,161,66,66,16,16,132,132,129,129,66,66,28,
28,180,180,129,129
230 DATA
74,74,28,28,52,52,161,161,74,74,16,16,132,132,129,129,66,66,1
8,18,4,4,161,161
240 DATA
66,66,30,30,52,52,161,161,74,74,28,28,180,180,129,129
250 POKE 756,A
260 RETURN

```

10: El inicio del programa se encuentra en las líneas inferiores, para facilitar la rapidez de ejecución de la parte activa.

100: P\$ recibe el gráfico de pantalla en forma de string, cubriendo con ello 22 líneas de modo con 40 caracteres cada una. C\$ recoge el juego de los caracteres utilizados como elementos de imagen.

110: Llena P\$ con 880 rombos (caracteres numéricos) según el método expuesto en el programa ADR135A.BEC. Aquí también puede colocar cualquier otro carácter contenido en C\$ o secuencias de los mismos. Algunos de los caracteres de nueva creación contienen pocas partes en COLOR 3. Si llama estos caracteres en su forma inversa obtendrá el cuarto -o quinto, contando el fondo- color.

120 a 150: Reservan memoria para el nuevo juego de caracteres. En este caso no se precisa una zona muy grande de memoria, ya que sólo se confeccionan once nuevos caracteres.

200: Los DATA de los nuevos caracteres. Siguiendo el orden del código interno, el espacio está en lugar cero seguido de exclamación y comillas. Puesto que las comillas no pueden aparecer en un string, al menos en el BASIC del ATARI, en este caso se ha prescindido de los tres primeros caracteres. Pero los once caracteres siguientes sirven de elementos de imagen. Por ello la línea 150 lee catorce datos ($14*8=112$) con el bucle FOR-NEXT.

250: Cambia el puntero CHBAS y

260: después empieza el programa.

20 y 30: Aunque el contenido del gráfico de pantalla haya sido depositado en forma de caracteres en un string interminable, podemos actuar sin embargo como si tratásemos con posiciones de pantalla, designados habitualmente con X e Y. Puesto que el string se muestra (PRINT) en pantalla

comenzando siempre en el mismo lugar, cada elemento por separado también se emite en el mismo lugar de la pantalla. La función de P\$ aquí no es más que la de memoria de pantalla en la que se almacenan números de identificación, que servirán para realizar disposiciones gráficas de color más complejas, a saber los caracteres creados por nosotros mismos, y emitirlos en la pantalla.

De igual manera pueden imaginarse las funciones de la memoria de pantalla en el modo carácter. En las celdas de memoria se anotan los valores de los caracteres con su código interno, que servirán después para leer en el juego de caracteres ROM los bits tipo a representar en la pantalla de TV.

La gran ventaja del método del string estriba en que no hay ningún problema con la memoria a reservar. Lo que los elementos de imagen definidos muestran en la pantalla también se podría conseguir de forma parecida en GRAPHICS 15 (XL), pero sólo con penosos PLOTS aislados que requerirían además el consiguiente tiempo de ejecución. En este caso se rellenaría con cada elemento (carácter) una superficie de ocho por ocho puntos de imagen (4 por ocho pixels). La limitación consiste en que el gráfico de pantalla sólo puede componerse a partir de muestras prefabricadas al igual que un "collage". Sin embargo, al estudiar este programa también verá el tamaño tan minúsculo de estas piezas y que cada pixel se pierde en esta muestra.

El método del string simplifica pues la creación de los contenidos gráficos de colores de la pantalla, ocupando poca memoria y trabajando a velocidad relativamente elevada. Al tener que definir 128 elementos de imagen, se permite definir una variada caja de construcciones de piezas gráficas, que nos servirá para componer rápidamente por ejemplo un paisaje o un calidoscopio o cualquier otra cosa.

Estas dos líneas de programa determinan pues posiciones aleatorias de pantalla X e Y, tal como podría suceder también en otra forma de representación, como por ejemplo

GRAPHICS 3.

40: Aquí se extrae un número aleatorio entre 0 y 10, que deberá escoger más tarde el elemento creador.

50: J calcula la posición dentro del string a partir de X e Y, tal como se determinaría la celda de la memoria de pantalla a partir de X e Y, es decir valor de línea (Y) multiplicado por la cantidad de caracteres por línea (40) más valor de columna (X). Puesto que trabajamos con un string y que el primer carácter de un string no tiene valor de posición 0 sino 1, habrá que sumarle además un 1 al valor hallado.

Lo mismo vale para el número aleatorio Z. Un string no tiene ningún carácter número 0.

60: El carácter que se encuentra en la posición J calculada (P\$(J,J)) de P\$ es sustituido por el carácter Z hallado aleatoriamente, que se encuentra en la posición Z de C\$.

70: Emite (PRINT) P\$ en la pantalla comenzando por la esquina superior izquierda, iniciando después una nueva ejecución del programa.

Si el objetivo de un programa de este tipo tan sólo consiste en mostrar una imagen variable de colores en la pantalla, no será necesario introducir en el string continuamente los elementos de imagen determinados de forma aleatoria, ni imprimir tampoco el P\$ completo en cada pasada. Se puede conseguir el mismo efecto al señalar con POSITION la posición de pantalla determinada aleatoriamente y emitiendo (PRINT) el elemento de imagen C\$(Z,Z) en ella.

El método aquí presentado va almacenando continuamente el actual contenido de imagen, por lo que éste es independiente de su representación en el monitor. El cambio también podría continuar mientras se muestre alguna otra cosa en la pantalla. El programa también podría conmutar espontáneamente entre P\$ y otros strings con diferentes contenidos de imagen, resolviendo de esta forma elegante y ahorrativa (de memoria) el problema de un rápido cambio de escena en un modo gráfico de alta resolución.

LABEL - ABECEDARIO

APPMHI	14	GRACTL	53277	TMPCHR	80
ARGOPS	128	GRAFM	53265	TMPCOL	679
ATACHR	763	GRAPFO-3	53261	TMPLBT	673
ATTRACT	77	HATABS	794	TEMPROW	696
AUDC1	53761	HITCLR	53278	TRAMSZ	6
AUDC2	53763	HLPFLG	732	TSTAT	793
AUDC3	53765	HOLDCH	124	TSTDAT	7
AUDC4	53767	HOLD1	81	TXTCOL	657
AUDCTL	53768	HPOSMO-3	53252	TXTMSC	660
AUDF1	53760	HPOSP0-3	53248	TXTOLD	662
AUDF2	53762	ICAX1Z-6Z	42	TXTROW	656
AUDF3	53764	ICBALZ/HZ	36	VBREAK	518
AUDF4	53766	ICBALLZ/HZ	40	VDELAY	53276
BFENLO/HI	52	ICCOMT	23	VDSLST	512
BITMSK	110	ICCOMZ	34	VIMIRQ	534
BOOT?	9	ICDNOZ	33	VINTER	516
BOTSCR	703	ICHIDZ	32	VKYBD	520
BPTR	61	ICPTLZ/HZ	38	VNDT	132
BRKKEY	17	ICSTAZ	35	VNTP	130
BRKKY	566	INBUFF	243	VPRCED	514
BUFADR	21	INITAD	738	VSERIN	522
BUFCNT	107	INSDAT	125	VSEROC	526
BUFRFL	56	INTEMP	557	VSEROR	524
BUFRLO/HI	50	INTRVEC	552	VTIMR1-4	528
BUFSTR	108	INVFLG	694	VVBLKD	548
CASFLG	783	IOCBO-7	832	VVBLKI	546
CASINI	2	IRQEN	53774		
		KEYDEF	121	VVTP	134
CASSBT	75	KEYDEL	729	WARMST	8
CBAUD	750	KEYDEL	753	XMTDON	58
CDTMA 1-2	550	KEYREP	730		

CDTMF3-5	554	LCONT	563
CDTMV1-5	536	LINBUF	583
CFB	570	LINZBS	0
CH	764	LMARGIN	82
CH1	754	LOGCOL	99
CHACT	755	LOGMAP	690
CHACTL	54273	LOMEM	128
CHAR	762	LPENH/V	564
CHBAS	756	MEMLO	743
CHBASE	54281	MEMTOP	144
CHKSNT	59	MEMTOP	741
CHKSUM	49	NEWCOL	97
CIX	242	NEWROW	96
CKEY	74	NOCKSM	60
COLAC	114	NOCLIK	731
COLBK	53274	OLDADR	94
COLCRS	85	OLDCHR	93
COLDST	580	OLDCOL	91
COLORO-4	708	OLDROW	90
COLPFO-3	53270	OUTBUFF	128
COLPMO-3	53266	PADDLO-7	624
COLRSH	79	PBPNT	29
CONSOL	53279	PBUFSZ	30
COUNTR	126	PCOLRO-3	704
CRETRY	54	PMBASE	54279
CRITIC	66	PRIOR	53275
CRSINH	752	PRNBUF	960
DAUX1-2	778	PTABW	201
DBSECT	577	PTEMP	31
DBUFLO/HI	772	PTIMOT	28
DBYTLO/HI	776	PTRIGO-7	636
DCB	768	RADFLG	251
DCOMND	770	RAMLO	4
DDEVIC	768	RAMSIZ	740
DEGFLG	251	RAMTOP	106
DELTAC	119	RANDOM	53770

DELTAR	118	RECVDN	57
DINDEX	87	RMARGN	83
DLISTL/H	54274	ROWAC	112
DMACTL	54272	ROWCRS	84
DMASK	672	ROWINC	760
DOSINI	12	RTCLOK	18
DOSVEC	10	RUNAD	736
DRETRY	55	RUNSTK	142
DRKMSK	78	SCRFLG	699
DSKFMS	24	SDLSTL	560
DSKTIM	582	SDMCTL	559
DSKUTL	26	SHFAMT	111
DSPFLG	766	SHFLOK	702
DSTAT	76	SOUNDR	65
DSTATS	771	SRTIMR	555
DTIMLO	774	SSFLAG	767
DUNIT	769	SSKTCL	562
DUNUSE	775	STACKP	792
DVSTAT	746	STARP	140
ENDPT	116	STATUS	48
ENDSTAR	142	STICKO-3	632
ERRSAVE	195	STMCUR	138
ESCFLG	674	STMTAB	136
ESIGN	239	STOPLN	186
FEOF	63	STRIGO-3	644
FILDAT	765	STRTFLG	1001
FILFLG	695	SWPFLG	123
FMZSPG	67	TABMAP	675
FREQ	64	TIMER1	780
FTYPE	62	TIMER2	784
GLBABS	736	TIMFLG	791
GPRIOR	623	TINDEX	659

ATASCII

Código que asigna un valor decimal entre 0 y 127 a cada uno de los caracteres disponibles en el ordenador. El código ATASCII es una modificación de la norma ASCII (American Standard Code for Information Interchange).

BCD

Se utilizan principalmente dos métodos diferentes para procesar valores numéricos. Con el método de números enteros se convierten simplemente los valores decimales en binarios. Estos datos pueden procesarse con mucha rapidez, pero ocasionan muchos errores de redondeo con mayor facilidad. El segundo camino consiste en la codificación binaria (Binary Coded Decimals). Trabajando con las constantes BCD de 6 bytes utilizadas por el ATARI, cada valor numérico ocupa una secuencia de 6 bytes. El primero recoge el signo y el exponente de cada valor; los cinco bytes siguientes contienen las cifras del número decimal transformadas por separado al sistema binario. Al precisar cuatro bits para representar las cifras 0 a 9, cada byte de BCD puede recoger dos cifras, de manera que las constantes BCD de 6 bytes puedan entender números con exactitud de diez cifras. Para ello se ocupa una zona de memoria relativamente amplia y los procesos de cálculo son bastante más lentos. (vea STARP; 140,141 = \$8C,\$8D)

Bit

Una cifra binaria (Binary digit) es la unidad más pequeña que el ordenador puede procesar. El sistema binario tiene base 2, es decir que sólo existen dos cifras, el 0 y el 1, y cada posición de un número binario representa una potencia de base 2. El ordenador considera las cifras 0 y 1 como estados de tensión eléctricos encendido y apagado. Un bit es la unidad de Información más pequeña imaginable, que contiene una decisión sí/no.

Bit tipo

Disposición de ocho bits en un byte, que con su estado de 0 ó 1 son considerados como tipo (muestra).

Byte

Los números binarios pueden alcanzar secuencias considerables de cifras, por lo que varios bits se agrupan en una unidad superior. Un byte tiene ocho bits. Ocho bits pueden representar valores decimales de 0 a 255. Un procesador de 8 bits, que actualmente todavía es el procesador estándar utilizado en ordenadores domésticos, procesa un byte entero (ocho bits en paralelo). La capacidad del procesador también se designa por palabra (word). Para un procesador de 8 bits, una palabra tiene un tamaño de un byte; para un procesador de 16 bits, una palabra consta de dos bytes, etc.

Byte de datos

El valor decimal (de 0 a 255) de un byte en contraposición al bit tipo.

Byte HI

Para representar valores numéricos mayores, como por ejemplo direcciones, el valor se descompone en las partes superior e inferior. El byte HI indica el múltiplo de 256 y el byte LO el resto del número: $HI * 256 + LO$. De esta manera pueden expresarse valores de 0 a 65535. Las dos partes siempre se memorizan en el orden LO, HI.

Byte LO

Ver byte HI.

CIO

Central Input/Output; rutinas centrales de entrada/salida que supervisan los procesos de entrada y salida.

DL

Display List; un programa en lenguaje máquina para el microprocesador ANTIC, que dispone de un propio juego de instrucciones. El DL determina la forma en que los datos son emitidos desde el ordenador hacia el monitor o TV.

DLI

Display List Interrupt; mientras que el haz catódico del tubo del televisor salta del margen derecho hacia el

comienzo de la línea siguiente (HBLANK = horizontal blank), se puede interrumpir el tratamiento del DL con el fin de procesar otras partes de programa en este corto espacio de tiempo.

DOS

Disk Operating System; un paquete software cargado del diskette cuya misión es la de coordinar la unidad de disco y el ordenador.

DUP

Disk Utility Package; software que maneja determinados comandos del DOS tales como COPY, etc.

EOF

End Of File; marca el final de un fichero.

EOL

End Of Line; marca el final de una línea lógica.

FMS

File Management System; parte del DOS que controla las operaciones de entrada/salida "D:".

FP

Floating Point; rutina matemática utilizada para procesar números decimales de coma flotante.

Gráfico PM

Player Missile Graphics; jugadores y proyectiles, una variante particular de sprites, objetos gráficos, independientes del contenido normal de la pantalla (GRAPHICS) en lo que a forma, color y movimiento se refiere.

I/O

Input/Output; entrada/salida.

IOCB

Input/Output Control Block; bloque de control de entrada/salida utilizado por el CIO para acceder a los periféricos.

Línea de modo

Unidad de trabajo para la representación gráfica de la pantalla. Según el modo gráfico escogido, una línea de modo puede ocupar 1, 2, 4, 8, 10 ó 16 líneas de TV. La instrucción ANTIC correspondiente determina el número de scan lines tratadas como línea de modo.

LMS

Load Memory Scan; apunta con un puntero de 2 bytes hacia aquella parte de la memoria donde se encuentran datos a procesar, como por ejemplo datos de imagen en el DL, e indica al ordenador que debe procesar estos datos.

Motor

Un programa motor (manejador) coordina el ordenador con los diversos periféricos tales como "E:"ditor, "K:"eyboard, "P:"rinter o "D:"iskette.

Nibble

Hace referencia a medio byte, tanto a la mitad inferior como a la superior de un byte. Al programar en lenguaje máquina se reúnen ocho bits para escribir en notación hexadecimal. Los números hexadecimales tienen base 16, se componen pues de 16 cifras (0 a 9, A, B, C, D, E, F). Cada posición de un número hexadecimal corresponde a una potencia de 16. Para representar una cifra hexadecimal se precisan cuatro bits, o sea medio byte o nibble. Un byte abarca valores de 00000000 a 11111111 en notación binaria, de 0 a 255 en decimal y de 00 a FF en hexadecimal.

OS

Operating System; el sistema operativo o de operaciones que controla las funciones del ordenador.

bytes HI de todas las direcciones de memoria marcan el principio de una página.

Pixel

Picture Element; elemento de imagen que conforma el contenido gráfico de una imagen de TV alimentada por el ordenador. Unidades de medida del tubo de televisión son líneas (scan lines) y puntos de imagen (color clocks; un color clock comprende dos puntos de imagen). Un pixel puede tener una altura definida por un número variado de líneas de pantalla y un ancho de varios puntos de imagen, dependiendo de la resolución de modo gráfico escogido.

Powerup

La conexión (powerup) da lugar a la ejecución de varias rutinas que preparan al ordenador para su funcionamiento. Por ejemplo en varios registros se escriben valores iniciales (default) y se comprueban determinados estados, como por ejemplo si está conectada una estación activa de diskettes.

Puntero

Vea vector.

RAM

Random Access Memory; memoria de acceso disponible o de lectura y escritura, que puede ser utilizada por el usuario para escribir sus programas BASIC. La memoria RAM sólo puede almacenar información mientras haya tensión en el dispositivo. Al desconectarlo se borra toda la memoria RAM.

Registro de sombra

Para permitir en BASIC el acceso a determinados registros hardware, se han creado registros de sombra en los que pueden depositarse los valores deseados. Una escritura en los registros hardware no tendría ningún efecto, puesto que cada 1/50 segundo se vuelve a escribir encima de muchos de ellos. El BASIC es pues demasiado lento para intervenir aquí directamente. Los valores de los registros de sombra son leídos según el impulso hertz de la máquina. Haciendo este rodeo se puede dar lugar a cambios permanentes.

ROM

Read Only Memory; Memoria de lectura anclada en chips de hardware, que contiene las informaciones necesarias del sistema. Gracias a la memorización hardware estas informaciones existen estructuralmente, es decir que permanecen al desconectar el flujo eléctrico.

SIO

Serial Input/Output; Rutinas encargadas de la entrada y salida de datos en serie.

Valor default

Valor por defecto.

VBLANK

Vertical Blank; espacio de tiempo del momento en que el haz catódico ha llegado a la esquina inferior derecha del tubo de televisión y vuelve a saltar hacia la esquina superior izquierda. En este corto instante se desconecta el haz catódico. En la norma PAL un VBLANK se produce cada 1/50 segundo.

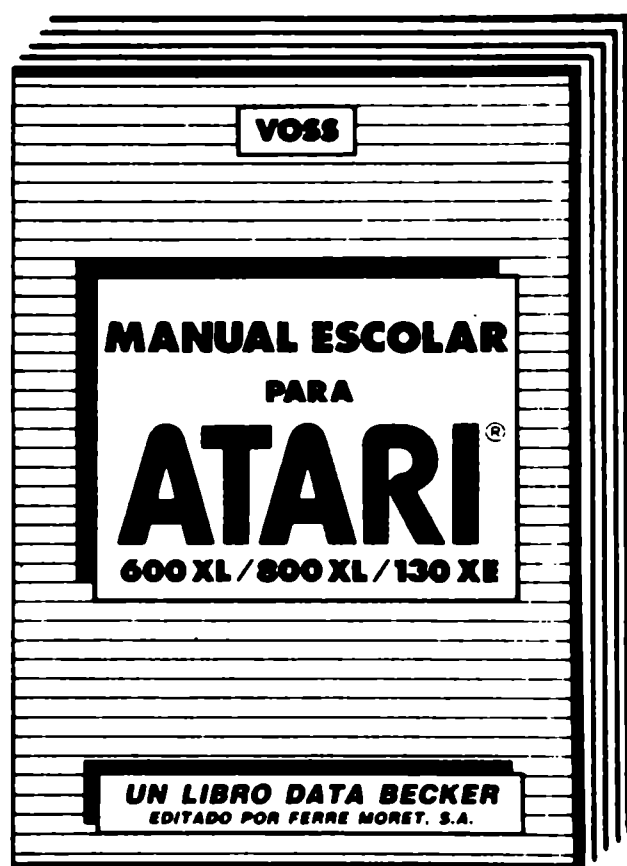
Vector

Registro que contiene una dirección determinada, como por ejemplo la dirección inicial de una zona de memoria o de una rutina de programa. Para apuntar hacia cualquier dirección, el vector precisa de dos bytes. Un vector de 1 byte contiene tan sólo el byte HI, es decir que sólo puede apuntar hacia el principio de una página.

Valor posicional

Cada cifra de un número obtiene un valor en función de su posición. En el sistema decimal cada posición corresponde a una potencia de base diez, o sea 1, 10, 100, etc. De la misma manera un bit tiene cierto valor por su posición dentro del byte, determinado por potencias de base dos. El bit 7 tiene valor posicional 2 elevado a $7 = 128$ decimal.

**Este libro se terminó de imprimir en el mes
de Junio de 1987 en la Planta Impresora
de Edicient S.A.I.C., cita en Mario Bravo 465/71
(1870) Avellaneda - Pcia. Buenos Aires**



**EL LIBRO ESCOLAR PARA
ATARI 600XL/800XL/130XE**

Muchos programas interesantes de soluciones de problemas y de aprendizaje, descritos de forma amplia y comprensible, y adecuados sobre todo para escolares. ¡Aquí el aprendizaje intensivo se convierte en una tarea divertida! Al margen de temas como los verbos irregulares, o las ecuaciones de segundo grado, un resumen corto de las bases del tratamiento electrónico de datos, y una introducción a los principios del análisis de problemas, completan este libro que debería obrar en posesión de cualquier escolar.

Voss

El libro escolar para ATARI 600XL/800XL/130XE
389 páginas, 2.800 .- ISBN 84-86437-12-1



**AVENTURAS
Y COMO PROGRAMARLAS EN EL
ATARI 600XL/800XL/130XE**

Jugar a aventuras con éxito y programarlas uno mismo - todo lo verdaderamente importante sobre el tema, lo contiene este guía fascinante que te lleva través del mundo fantástico de las aventuras. El libro abarca todo el espectro, hasta las más sofisticadas aventuras gráficas llenas de trucos, acompañándolas siempre de numerosos programas ejemplo. Sin embargo la clave - al margen de muchas aventuras para teclear - es un generador de aventuras completo, mediante el cual la programación de aventuras se convierte en un juego de niños.

Walkowiak

Aventuras - y como programarlas en el ATARI 600XL/800XL/130XE
284 páginas, 2.200 .- ISBN 84-86437-11-3



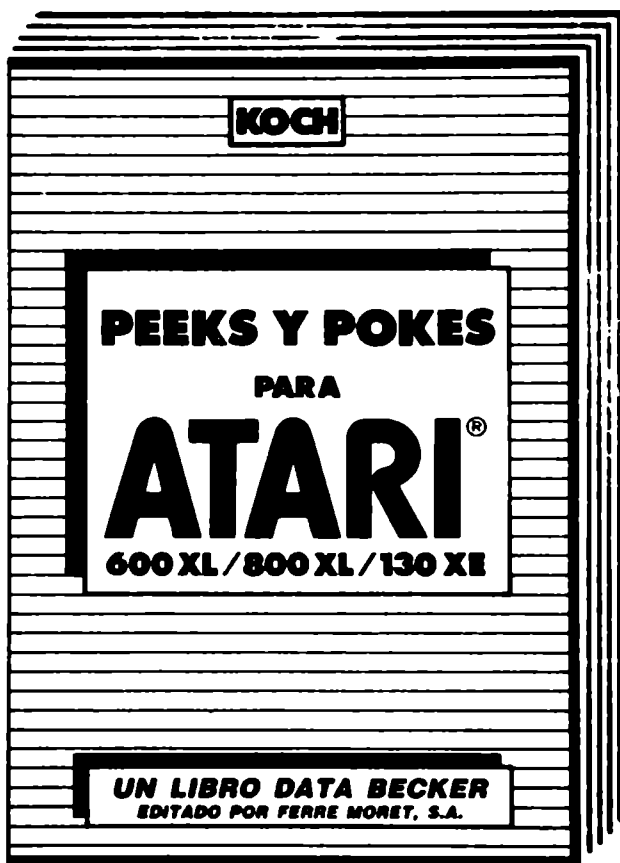
JUEGOS ESTRATEGICOS
Y COMO PROGRAMARLOS EN EL
ATARI 600XL/800XL/130XE

Una lograda introducción al sugestivo tema de los "juegos estratégicos". Desde juegos sencillos con estrategia fija a juegos complejos con procedimientos de búsqueda hasta programas con capacidad de aprendizaje - muchos ejemplos interesantes, escritos por supuesto de forma fácilmente comprensible. Con programas de juegos ampliamente detallados: NIM con un montón, bloqueo, hexapawn, mini-damas y muchos más.

Schneider

Juegos estratégicos - y como programarlos en el ATARI
600XL/800XL/130XE

181 páginas, 1600 .- ISBN 84-86437-14-8



PEEKs Y POKES PARA ATARI

Tan interesante como el tema, es el libro que explica de forma fácilmente comprensible el manejo de Peek's y Pokes importantes, y representa un gran número de Pokes con sus posibilidades de aplicación, incluyendo además programas ejemplo. Al lado de temas como lo son la memoria de la pantalla, los bits y los bytes, el mapa de la memoria, la tabla de modos gráficos o el sonido, también se detalla de forma magnífica la estructura del ATARI 600XL/800XL/130XE.

Koch

Peeks y Pokes para

ATARI 600XL/800XL/130XE

251 páginas, 2.200 .- ISBN 84-86437-13-X

Sello librero

Puesta al día de datos

EDITORIAL DATA BECKER S.A. mantiene vivo y amplía el contenido informativo de sus libros y programas, mediante el envío de un servicio de puesta al día, junto con una síntesis noticiosa de la actualidad y perspectivas de la realidad informática en lengua castellana.

Agradecemos cualquier sugerencia o crítica que desee formular y que nos ayude a mejorar las ediciones. Muchas gracias.

¿Qué añadiría?

.....

.....

¿Qué suprimiría?

.....

.....

Observaciones

.....

.....

Título del libro

.....

Nombre

.....

Dirección

.....

Tfno.

.....

Código Postal y Población

Provincia

.....

UN SERVICIO GRATUITO



Sello librero

Información y Pedidos

DATA BECKER S.A. cuenta con un amplio fondo de libros y Software y mantiene un servicio de información por correo sobre las novedades que edita.

Agradecemos nos indique los temas que representan para Vd. mayor interés.

Libros

ATARI

MSX

AMSTRAD

COMODORE

LENGUAJES

APPLE

SINCLAIR

IBM

SOFTWARE

Si está interesado en recibir alguno de estos servicios, rellene y envíe la tarjeta correspondiente; no necesita franqueo. Muchas gracias.

DESEO RECIBIR EL LIBRO

EL PROGRAMA

Adjunto cheque

Contra reembolso

.....

Nombre

.....

Dirección

.....

Tfno.

.....

Código Postal y Población

Provincia

.....

UN SERVICIO GRATUITO

**RESPUESTAS POSTALES
PAGADAS**

El franqueo será
pagado por el
Destinatario

DATA BECKER S.A.

**APARTADO ESPECIAL N° 4
1448 – SUCURSAL 48 (B)
BUENOS AIRES – ARGENTINA**

**RESPUESTAS POSTALES
PAGADAS**

El franqueo será
pagado por el
Destinatario

DATA BECKER S.A.

**APARTADO ESPECIAL N° 4
1448 – SUCURSAL 48 (B)
BUENOS AIRES – ARGENTINA**

EL CONTENIDO:

Este libro le explica de forma sencilla el manejo de las órdenes PEEK y POKE. Contiene un gran número de importantes «POKEs», con sus aplicaciones y muchos programas ejemplo. Además se describe muy bien la arquitectura del ATARI 600/800 XL / 130 XE.

Extracción del contenido:

- ATARI-BASIC-PEEK-POKE
- Juegos de números
- Desfile de bits
- Acercándonos a ROM y RAM
- Mapa de memoria
- Gráficos de misiles para juegos
- Sonido
- Lista de display
- Memoria de Pantalla
- Tabla de modos gráficos
- Juego de caracteres
y mucho más

ESTE LIBRO HA SIDO ESCRITO POR:

Karl-Heinz Koch, autor con años de experiencia en microcomputadores. Este ya es su segundo libro sobre ATARI, escrito de forma interesante y fácilmente comprensible.