La primera revista del mundo que enseña la programación de videojuegos

CODIGO JUEGO

975 pts
Canarias 1000 pts

Año 1, Número 4, Septiembre 1994

Para todo tipo de ordenadores personales

Cursos

- SONIDOS
- MIDI
- ENSAMBLADOR PC
- ENSAMBLADOR AMIGA
- ENSAMBLADOR ATARI
- LENGUAJE C
- PASCAL
- WINDOWS
- MSDOS
- TARJETA VGA

EL HARDWARE DEL AMIGA

La historia de los Videojuegos



NOVEDAD EL AMOS DE AMIGA

Aprende a hacer un juego para PC de principio a fin



Mookie

8 424094 009756 00004

Ajatel S.L.

CODIGO JUEGO

Editorial: AJATEL S.L.

Director: Ramón López

Subdirectores: Joan Esquerra
Miguel A. Lobato

Directora de arte: Maite López Vidal

Colaboradores: José Manuel Ginés
Manuel Gallego
José A. Brihuega Parodi
Eduard Sánchez Palazón
Roberto Cruz
Alfonso Salgado Cachafeiro
María Jesús Cano Rueda
Javier Laynez
Alberto García
Leopoldo Fernández Prieto

Fotografía: Manuel Gallego Palacios

Corresponsales USA: Luis Vazquez
Vicente Santos

Jefe de publicidad: Joan Esquerra
ASINFO: Teléfono/FAX: 956-813129

Subscripciones: Miguel L. Moreno
Teléfono/FAX: 956-840246

Filmación: Texto y Color
Paseo de las Delicias, 15
Teléfono: 956-302131

Imprime: ALTAMIRA
Carretera de Barcelona, Km 11200
28022 Madrid, Tel. 91-747 33 33

Disquetes: KAO
Tel. 93-7182287

Distribuye: DISPAÑA, S.L. S.C.
Avenida de Valdelaparra, 27
Edificio II, local 9
Alcobendas, Madrid
Tel. 91-6620602

ISSN: 1134-0975

Depósito Legal:CA-286/94

Código Juego
Apartado de Correos 114
11520 Rota, Cádiz
Teléfono/FAX:956-840246
Teléfono/FAX:956-813129

Código Juego no se hace necesariamente solidaria con las opiniones vertidas en los artículos firmados. Está prohibido cualquier tipo de reproducción por cualquiera que sea el medio, sin permido expreso del editor.

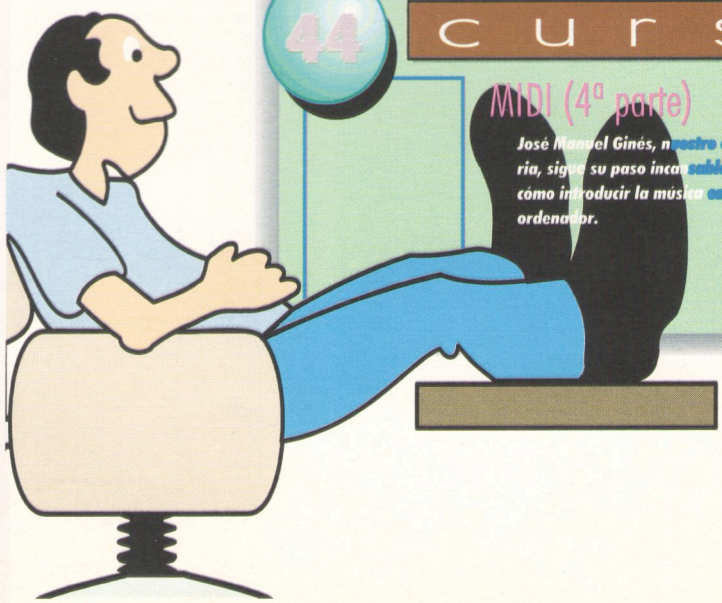
SUMARIO

6 curso
Ensamblador PC (4ª parte)
La cuarta entrega de este curso veterano, pero importantísimo en nuestra revista.

40 curso
Ensamblador Atari (4ª parte)
También los que poseen un ordenador Atari, encuentran en nuestras páginas la referencia técnica necesaria

42 curso
Ensamblador Amiga (2ª parte)
Hemos decido separar los cursos de ensamblador de Atari y Amiga aunque tengan el mismo microprocesador. Así unos y otros tendréis un curso más específico.

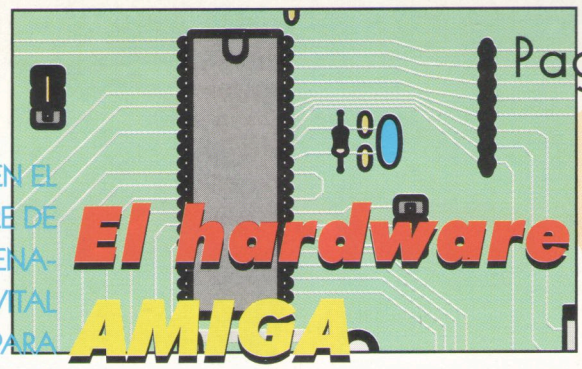
44 curso
MIDI (4ª parte)
José Manuel Ginés, nuestro experto en la materia, sigue su paso incansable para enseñaros cómo introducir la música en los juegos de ordenador.



Septiembre 1994

Año 1, número 4

Editorial 4
Rompecabezas informático 5
Carta del Lector 14
Tintilla 15
Así se hace un juego PC 16
La virguería 22
El libro del mes 24
El juego del mes 26
Curso lenguaje Pascal 28
La tarjeta VGA 31
Lenguaje C 36
Macroassembler 6.0 48
Curso de sonidos 50
La triquiñuela 52
El contestador 53
El buen SOFT 55
Así se hace un juego C 56
El MSDOS y los juegos 60
Herramientas 64
En el próximo número 67



Pag. 33

El hardware del AMIGA

CONOCER BIEN EL HARDWARE DE NUESTRO ORDENADOR ES DE VITAL IMPORTANCIA PARA SACAR EL MÁXIMO PARTIDO DEL MISMO. AHORA LE TOCA AL AMIGA.



Pag. 20

LA HISTORIA DE LOS VIDEOJUEGOS

ESTUDIAR COMO LOS VIDEOJUEGOS HAN EVOLUCIONADO A LO LARGO DE LOS AÑOS PUEDE SERNOS DE GRAN VALOR A LA HORA DE DECIDIR QUE TIPO DE PROGRAMA VAMOS A REALIZAR.

Nota de la redacción:
En el número 3 de Código Juego hubo dos errores lamentables que hicieron que no apareciese el texto en dos de los árticulos. Concretamente en "Gráficos 3D" y "Curso de Pascal". Subsanamos el érror en este número con la publicación del texto que no salió en su momento. Página 66.

EL JUEGO DEL MES



Editorial

Por:

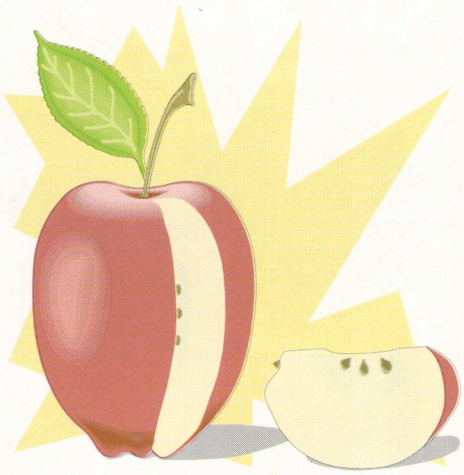
Ramón López

Las manzanas podridas

Yo me pregunto, ¿por qué siempre que algo bueno y nuevo nace enseguida le surjen detractores? Me refiero a esos seres neutros, que se me antojan descerebrados, sin lógica consciente ni madurez de pensamiento. Aquellos que pretenden ser Julio Verne y no llegan a ver más allá de sus gordas narices. Aquellos que ven la informática como una diosa regordeta y dañina, que con su paso acelerado y firme se acerca poco a poco a nuestras vidas, ofreciéndonos un caramelo goloso para la sociedad actual; la comodidad. Es cómodo utilizar un ordenador y que éste te regule el funcionamiento las cosas de la casa, te lleve la contabilidad, te ayude a escribir un libro, , te deje ver las fotografías de tu último viaje a la costa brava, te auxilie con tu problema médico y que encima te permita dar un paseo por el camino de Santiago sin moverte de tu casa.

Pero la comodidad y las otras muchas ventajas que ofrecen nuestras amigas las máquinas, tienen un precio, y alto. La mecanización resalta la inutilidad de hombre ante ciertas tareas rutinarias y repetitivas que van en contra de nuestra naturaleza equivocativa y despistada, la cual se rompe en mil pedazos cuando colisiona con la perfección y exactitud de los ordenadores, que los únicos errores que cometen, encima, son los que nosotros les hacemos cometer. Una auténtica paradoja.

Y a ello se agarran los que quieren hundir, la ya imparable, ascensión de la informática y los ordenadores personales. Estas personas se imaginan los años venideros como el escenario de la película Terminator, donde las computadoras son capaces de replicarse y pensar, ya no como los humanos, sino incluso mejor y con más malicia y afán exterminador. Robots creados para asesinar y para dejar sin trabajo a los hombres. No se sabe que es peor.



Pero estos seres nulos de entendimiento nunca miran al lado blanco y esperanzador. Gracias a la informática, hoy en día, muchas personas impedidas pueden conducir un vehículo, moverse en una silla de ruedas, comunicarse verbalmente, y en definitiva, ser más personas. La utilización exaustiva de los ordenadores en la medicina ha hecho posible pruebas de diagnóstico como la resonancia magnética (MRI) o la tomografía axial (CAT SCAN) sean concluyentes a la hora de encontrar remedio contra el cancer y otras muchas afecciones de difícil diagnóstico. Sin la informática, la investigación científica andaría a pasos de tortuga con el consiguiente atraso que ello supondría. Tenemos que agradecer muchas cosas a "las cajas inteligentes". Más de lo que creemos.

El caso es que ahora, para colmo de males, se le hecha la culpa a los juegos de rol de que ciertos individuos asesinen a sus semejantes como consecuencia de no saber poner un tope entre la realidad y la ficción. Realidad que, por otro lado, siempre ha sido un ingrediente primordial del cine y no por ello se ha prohibido la realización de películas violentas.

No creo que debamos meter en un mismo cesto todas las manzanas, sólo algunas están podridas, pero gracias a Dios son las que menos.

——— Me decís de todo en vuestras cartas. Os gustarían más páginas, más cursos, más apartados, un formato de revista más grande, que hablásemos de las consolas -reinas indiscutibles de los juegos electrónicos- que abaratemos el precio, que regalemos más de un disquete, etc, y el caso es que tenéis razón.

Por un lado, y como director de esta publicación, no tengo más remedio que coger el latigo de la estrechez monetaria y soltar, de vez en cuando, algún que otro zarpazo por culpa del presupuesto de nuestra publicación. Alguien tiene que hacerlo, ¡qué le vamos a hacer! La verdad es que nos gustaría poner disquete siempre, pero en ocasiones no nos es posible. Aunque no ocurrirá así muy a menudo, os lo prometo. Lo que sí os digo es que, cuando lo hagamos, vamos a preocuparnos de poner en nuestros disquetes el suficiente material para explicaciones posteriores. De esa forma siempre tendremos programas que estudiar conjuntamente con el desarrollo de la revista.

Ante tanta expectativa, y porque nos debemos a vosotros, os puedo adelantar una cosa en firme. La revista aumentará de tamaño en un plazo no muy lejano, concretamente antes de fin de año. Daremos espacio entonces a vuestras solicitudes y, como no, nuestras secciones ya consagradas irán ganando alguna que otra página más.

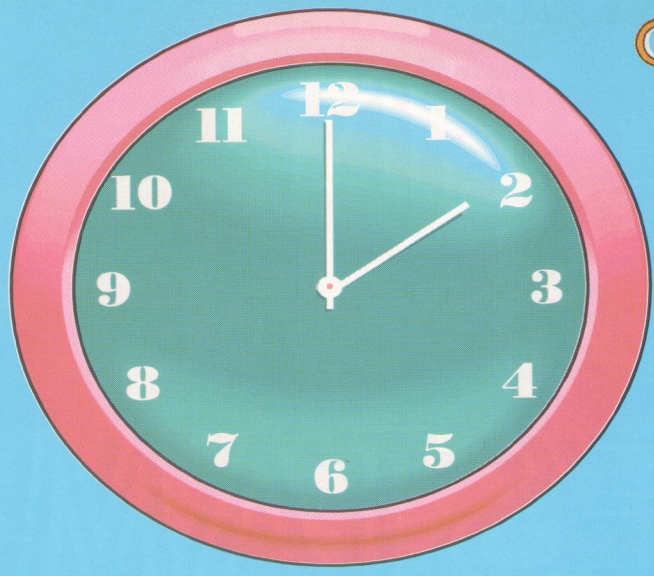
Nuestro objetivo se ha cumplido. Queríamos enseñar y a la vez divertir y por las conclusiones que hemos sacado de vuestras cartas y llamadas, llegamos a vosotros con agrado y de manera que os entretenéis y aprendéis. Nos alegramos de que así sea. Seguiremos, pues, en la misma línea de trabajo.

Este mes abrimos una sección interesante. Se trata de una historia sobre los videojuegos con la que pretendemos que conozcáis como han avanzado y mejorado estos con el transcurso de los años. Los juegos para ordenador son relativamente jóvenes, pero no por ello dejan de tener ya una dilatada historia de cambios, tanto en la forma de hacerlos como en la calidad que presentan. En esta sección, durante algunos meses, encontraréis una información interesante a la hora de elegir que tipo de juego os gustaría desarrollar, en incluso os podéis apoyar en estos datos cuando queráis averiguar si un determinado tipo de videjuego ha salido al mercado con anterioridad, para saber si vuestra idea es original o no.

Espero que os guste este número y quedo en espera veros en el siguiente. Un abrazo, de éste, vuestro director.

Ajatel S.L.

CODIGO JUEGO



Ya sólo quedan 2 meses

para el final

EL GRAN CONCURSO DEL ROMPECABEZAS INFORMATICO

Las bases del concurso son muy fáciles. Cada mes, y desde el número 1 de Código Juego se publica una pregunta cuya contestación es siempre un número de una única cifra. Cuando llegemos a la sexta pregunta, los seis numeros resultantes de éstas formarán un teléfono secreto.

El primer concursante que llame a este número en el día indicado en las bases que se explicaron en nuestra primera revista, recibirá en su domicilio un 486 DX2, totalmente gratis.

La pregunta de este mes:

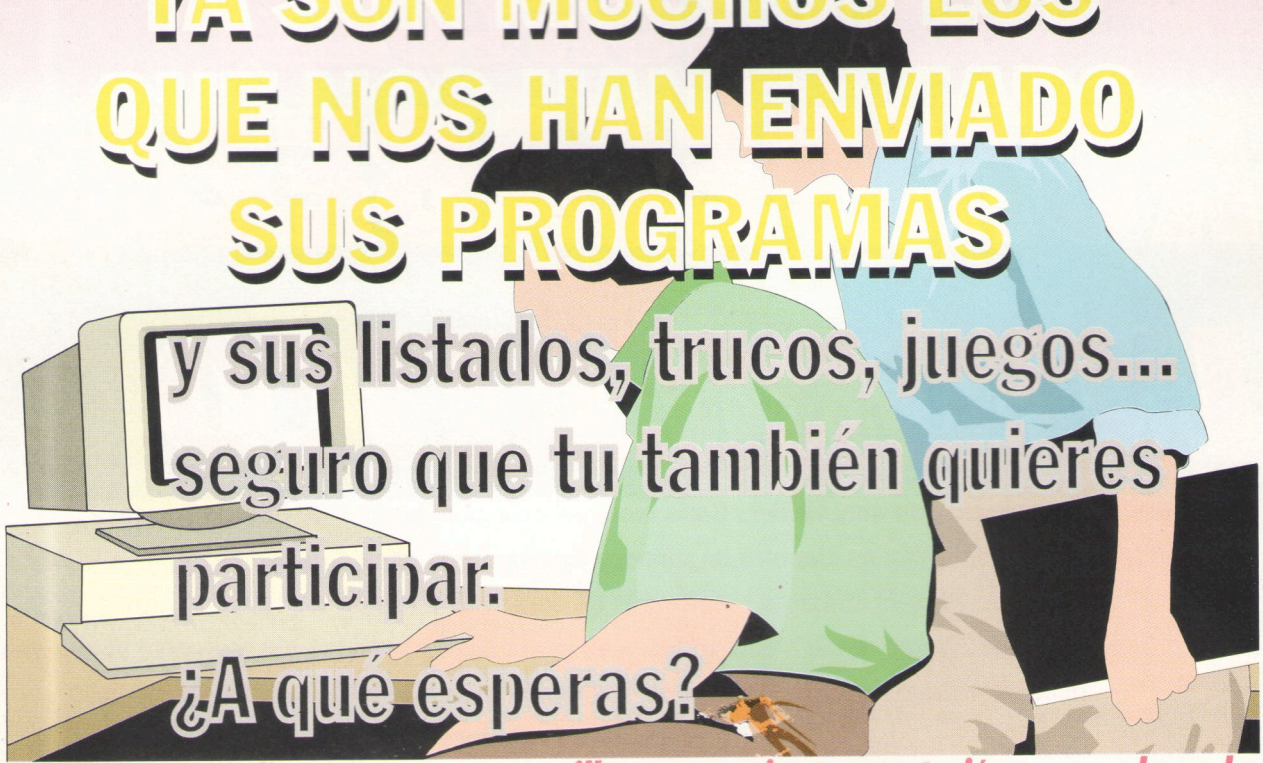
La memoria convencional de un PC esta limitada a una cierta cantidad de kilobytes, pero ¿sabes cuántos son?. Suma este dato a el número resultante de las operaciones del mes anterior y multiplícalo por 2. Obtendrás un número de 4 cifras con el que jugaremos el mes próximo.

EN

CODIGO JUEGO

TE ESTAMOS ESPERANDO

YA SON MUCHOS LOS QUE NOS HAN ENVIADO SUS PROGRAMAS



y sus listados, trucos, juegos...
seguro que tu también quieres participar.
¿A qué esperas?

Participar en Código Juego es tan sencillo como enviar tu aportación en un sobre a la dirección que detallamos abajo. Si tu programa es elegido contactaremos contigo.

Código Juego
Ajatel S.L.
Apdo. 114, 11520, Rota
Cádiz

Ajatel S.L.

COMO SI FUESEMOS DIMINUTAS HORMIGAS NOS VAMOS APODERANDO DE NUESTRO MICROPROCESADOR PARA QUE HAGA LO QUE NOSOTROS QUEREMOS. TODO ELLO GRACIAS A QUE VAMOS CONOCIENDO EL LENGUAJE ENSAMBLADOR, EL LENGUAJE DE LOS PROGRAMADORES PROFESIONALES DE VIDEOJUEGOS. HOY, ESTUDIAMOS A FONDO LA FORMA EN QUE EL MICROPROCESADOR SEGMENTA LA MEMORIA PARA PODER TRABAJAR CON ELLA.

Por:

Ramón López



CURSO DE LENGUAJE ENSAMBLADOR PARA PC

4ª PARTE

El mes anterior aprendimos muchas cosas útiles. Mover datos dentro de la memoria, de ésta hacia el microprocesador, de éste a la memoria o hacia los puertos de comunicaciones con los periféricos o con otros chips de apoyo, se puede decir que lo es todo en la programación en ensamblador. Los datos son como pequeñas gotitas de oro movidas por hormigas muy trabajadoras. Sin ellos, el ordenador no es nada, no tiene vida. Por ello, aprender bien desde un principio cuales son las instrucciones con las que contamos para traspasar la información entre los diferentes componentes internos de nuestro ordenador, toma una vital importancia.

Vimos, que mover implica tener un origen, de donde tomamos el dato, y un destino, hacia donde movemos o donde almacenamos dicha información. En realidad MOVE no implica borrar el dato del origen, o para que lo entendáis mejor, poner "a cero" ese registro o posición de memoria. Se realiza en realidad una copia, con lo que al finalizar la operación de traspaso, tenemos el mismo número en el lugar de origen y en el de destino.

lenguaje Ensambl

¿Para qué mover datos?, muy sencillo. Suponed por un momento que tenemos en el registro AX un dato importante que no queremos perder, pero que necesitamos utilizar dicho registro para una nueva operación. Como sabemos, al cargar el registro con el dato entrante el contenido previo se pierde. ¿Qué hacer? Dos cosas podemos. Una sería mover el dato hacia la memoria y almacenarlo allí. Otra podría ser pasárselo a un registro que no vayamos a utilizar por el momento. Como véis, estas que acabo de describir serían dos de las utilidades inmediatas de la orden MOVE, pero no la únicas ni las más importantes.

Tenemos, pues, la necesidad imperiosa de traspasar y almacenar la información importante para usos futuros.

Pero vamos a estudiar hoy algo todavía más importante, si no de cara al microprocesador, si para el programador.

Hasta ahora hemos visto como la CPU trabaja con los números. Hemos estudiado la notación binaria, que, por así decirlo, es la más rudimentaria, pero la que utiliza el ordenador y sin la cual éste, y todos, los artículos de esta revista no tendrían sentido puesto que los ordenadores no hubiesen aparecido jámas.

Pero nos encontramos con un problema. Si la notación decimal es críptica y enrevesada y difícilmente comprensible por nosotros, los humanos, y la notación decimal de aleja demasiado de lo que sería la comprensión del microprocesador, ¿cómo vamos a trabajar con los número entonces? La solución está en la notación hexadecimal.



Gracias al lenguaje ensamblador nuestros juegos consiguen toda la rapidez que necesitamos a la hora de mover gráficos por la pantalla.

NOTACION HEXADECIMAL

Ya hemos apuntado algunas notas importantes sobre este sistema de numeración en artículos anteriores de este curso, pero vamos a atar cabos para que lo entendáis perfectamente.

Trabajar con unos y ceros es una lata. Poner un 1 por un 0 ó viceversa, es tan fácil como despistarse un segundo en la confección de un programa y más cuando vamos a desarrollar un proyecto grande. Normalmente, no trabajamos con los bits de forma individual o separada sino que los agrupamos en paquetes de ocho a los que denominamos BYTES y que ya vimos en el número anterior.

Como los primeros microprocesadores trabajaban con cifras binarias en grupos de 4 o nibbles, los números decimales que se podían representar iban de 0 a 15, es decir 16 en total. Hacía falta un sistema de número que tuviese 16 cifras. Si el que tenía 10 cifras de llamó DECIMAL y el que tenía dos se denomino BINARIO, convinando el prefijo griego HEX (seis) y la palabra latina DECEM (diez) se acuño el término HEXADECIMAL o en base 16.

El problema surgió a la hora de escoger la representación gráfica de esos números. Si se utilizaban los dígitos del 0 al 9,

ador para PC

¿cómo representar los del 11 al 15 con una sola cifra? Esto se solucionó utilizando letras en lugar de números. Así, tenemos que el diez es la A, el 11 es la B, el 12 es la C, y así sucesivamente hasta el 15 que es la F. !A qué ya os van sonando un poco más los números como 12CD, 56FC y 34AB. ¿No?, bueno, continuemos un poco más.

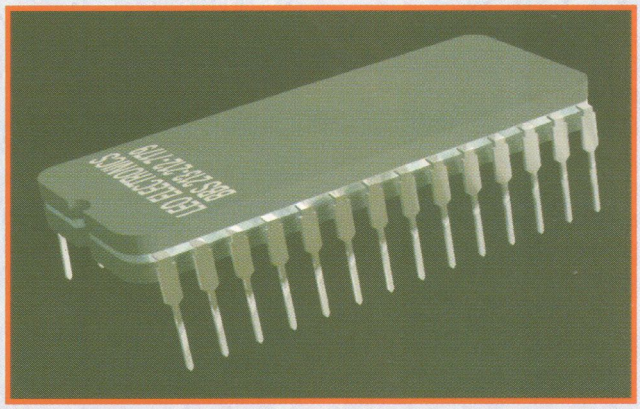
la figura 1, os muestra las equivalencias entre los tres sistemas de numeración que hemos visto ya.

Los valores de las posiciones hexadecimales son, como en los demás, el resultado de elevar la base por el lugar que ocupan. Ver figura 2.

Los números hexadecimales se utilizan para todo en lenguaje ensamblador, y aunque también podemos echar mano de las otras dos notaciones, es aconsejable "pensar en hexadecimal", ya que las posiciones de memoria, interrupciones, puertos, datos, código máquina, etc, se representan en este cómodo sistema numérico, que aunque difícil de entender y dominar en un principio, es mágnifico cuando lo llegamos a entender.

Ello se debe principalmente a la memoria y la forma en que es segmentada por el microprocesador. Cuando intentamos entender este hecho sin las "gafas hexadecimales" podemos volvernos locos. Representar una posición de memoria como 53829:35282 nos llevará definitivamente al caos, sin embargo la limpieza de la notación hexadecimal, 2345:7863, y su equivalencia total con el cerebro de la máquina, el microprocesador, hará que nos paseemos por la memoria como Pedro por su casa.

La memoria lo es todo para el microprocesador. En ella se guardan los programas cuando se cargan desde el disco para que puedan ser ejecutados y además es el lugar donde se almacenan los datos resultado de las operaciones que éste realiza.



ALGUNOS EJEMPLOS

Tomemos el número hexadecimal 2Dh. La "h" ya sea en mayúsculas o minúsculas, indica que se trata de un número en esa notación. Se le añade para que no existan confusiones con la notación decimal que también utiliza cifras comunes. Por ejemplo, los números 23 decimal y 23h son diferentes, ya que el segundo vale en realidad 35 en decimal. Por ello, no olvidéis nunca poner la H. También, y por costumbre se suele añadir esta letra identificativa a los números que van del 0 al 9, incluso cuando coinden en valor con sus correspondientes decimales. Así tenemos 0h y 9h. Ello hace que no nos confundamos con las notaciones y para que indiquemos que seguimos hablando en hexadecimal en todo momento.

Como deciamos, el número 2Dh nos va a servir de ejemplo para explicar como se traduce éste a notación binaria y decimal.

El paso de hexadecimal a binario es simple.

Sabemos que una cifra hexadecimal (en nuestro ejemplo el 2 o la D) equivale siempre a 4 bits. En otras palabras, con cuatro bits podemos representar los números hexadecimales que van desde 0h a Fh, como hemos podido ver en la figura 1. Cuando tenemos una cifra hexadecimal de 2 dígitos, automáticamente sabemos que como resultado de la traducción a binario vamos a obtener 8 bits. Los 4 primeros bits de este byte (empezando como siempre de derecha a izquierda) serán los que representa la primera cifra hexadecimal (Dh en nuestro ejemplo) y los cuatro bits siguientes, o nibble alto, serán los que se deriven de la segunda cifra hexadecimal (2h en nuestro ejemplo). Si la

Binario	Hexadecimal	Decimal
0000	0	1
0001	1	2
0010	2	3
0011	3	4
0100	4	5
0101	5	6
0110	6	7
0111	7	8
1000	8	9
1001	9	10
1010	A	11
1011	B	12
1100	C	13
1101	D	14
1110	E	15
1111	F	16

representación binaria de Dh es 1011b y la de 2h es 0010 el byte correspondiente a 2Dh es 00101011b. ¿Está claro, verdad?. Cuando se trate de 4 cifras hexadecimales, por ejemplo, 24F3h ocurrirá exactamente lo mismo. Siguiendo el orden, 3=0011, F=1111, 4=0100 y 2=0010, luego 24F3h=0010010011110011, y así sucesivamente con todas las palabras, dobles palabras y cuádruples palabras que queramos.

El sistema hexadecimal nos permite también hacer una traducción de un número a su equivalente binario con gran rapidez, cosa que nos será muy útil más adelante, como veréis en su momento.

En definitiva, trabajamos en binario, decimal y hexadecimal a nuestro antojo representativo, pero el microprocesador sigue digeriéndo su plato favorito; los unos y los ceros.

LA MEMORIA

Ya hemos hablado en muchas ocasiones de la importancia de la memoria y de su papel preponderante, pero ¿cómo trabaja el microprocesador con élla?

En capítulos anteriores, hemos visto que el microprocesador cuenta con unas zonas especiales donde almacena los datos para trabajar con ellos de manera más rápida que si los tuviese que almacenar en la "lejana" y lenta memoria. Estoy hablado de los **registros**.

Los registros son exactamente igual que una celdilla de memoria, sólo que dentro del microprocesador. Por ello, tienen la misma estructura que éstas, es decir bytes o palabras, depende si los tratamos como 8 o 16 bits. El registro AX, por ejemplo, es de 16 bits números del 0 al 15. Lo mismo ocurre con el registro CS, con DS y con DI, etc.

Lo que ocurre es que no todos los registros cumplen las mismas funciones. En realidad, podemos utilizarlos como se nos antoje (con alguna que otra restricción), pero si seguimos unas normas educadas nos irá mejor.

En el número 2 de CJ, os presentamos una tabla donde los registros aparecían subdivididos en diferentes grupos; datos, segmento, índice, etc. Ello se debe a que para diferentes tareas el microprocesador utilizará diferentes registros. Las funciones de cada uno de ellos las iremos viendo paso a paso, siendo la que nos interesa hoy la de los registros de segmento.

¿Qué quiere decir esto de segmento? Imaginaros una línea enorme. Cuando la dividimos en partes de igual longuitud llamamos a cada una de ellas segmento de la línea. Pues lo mismo ocurre con la memoria.

Ya sabemos que la memoria es un montón de "cajas de cerveza" (como vimos en su momento) alineadas y etiquetadas de 0 a XXXX (última posición de memoria disponible). Pues bien, el microprocesador no puede trabajar con toda la memoria a la vez. Sí, aunque os parezca extraño, la CPU es incapaz de controlar la memoria en su todalidad, por lo que la divide en partes iguales denominadas segmentos, al igual que ocurría con la línea del ejemplo anterior.

Para entender el porqué vamos a hacer un alto en el camino para explicar una cosa antes.

Los registros de segmento son cuatro: CS o segmento donde se encuentra el código o programa ejecutable, DS donde están los datos del programa, SS que es donde está la pila y ES el segmento extra de datos.

Hemos visto que estos segmentos son de 16 bits es decir que pueden contener números decimales que van desde 0 hasta 65565, o en hexadecimal 0000h hasta FFFFh. El microprocesador utiliza estos registros como punteros o indicativos de comienzo del las partes de nuestro programa. Por ello, siempre utilizará CS para saber donde se encuentra nuestro código ejecutable dentro de toda la memoria del PC. Pero, si la memoria es tan grande ¿cómo voy a trabajar con ella si el microprocesador sólo puede "ver" 65536 posiciones a la vez?

Esta es la gran duda y una de las materias más difíciles de entender para los programadores noveles.

Hemos visto que el microprocesador del PC utiliza un registro especial para que le indique en todo momento cual es la siguiente instrucción a ejecutar. Se trata de IP. Pues bien de la conjunción de CS y de IP obtenemos una dirección física dentro de la memoria que siempre será la celdilla de memoria donde está la instrucción que se ejecutará a continuación. Esto se anota de la siguiente forma:

CS:IP

Si introducimos el valor 0000h en CS y el valor 0001 en IP tendremos que la instrucción localizada en 0000:0001 será la próxima en ejecutarse.

Una buena pregunta sería, ¿A cuántas direcciones de memoria podemos acceder con CS igual a 0000h? Si CS=0000h e IP puede valer entre 0000h y FFFFh sacamos en conclusión que el número de celdillas a las que podemos acceder es de 65536. Pero, ¿qué ocurre si CS=0001h? De nuevo el número de celdillas es de 65536, pero ahorra podemos acceder a 16 celdillas que antes estaban fuera de nuestro alcance.Vaya lio, esto no hay quien lo entienda. Por eso vamos a hacer una parada y reflexionar lo que hemos visto.

Según la arquitectura interna de registros del microprocesador del PC, ésta siempre obliga a la utilización de los registros de segmento para indicar donde comienzan las instrucciones y datos de nuestro programa. Si situamos el comienzo de nuestro segmento en una determinada posición de memoria sólo podremos movernos 65536 bytes a partir de éste.

El lio lo podéis ver un poco más claro si miráis en

ador para PC

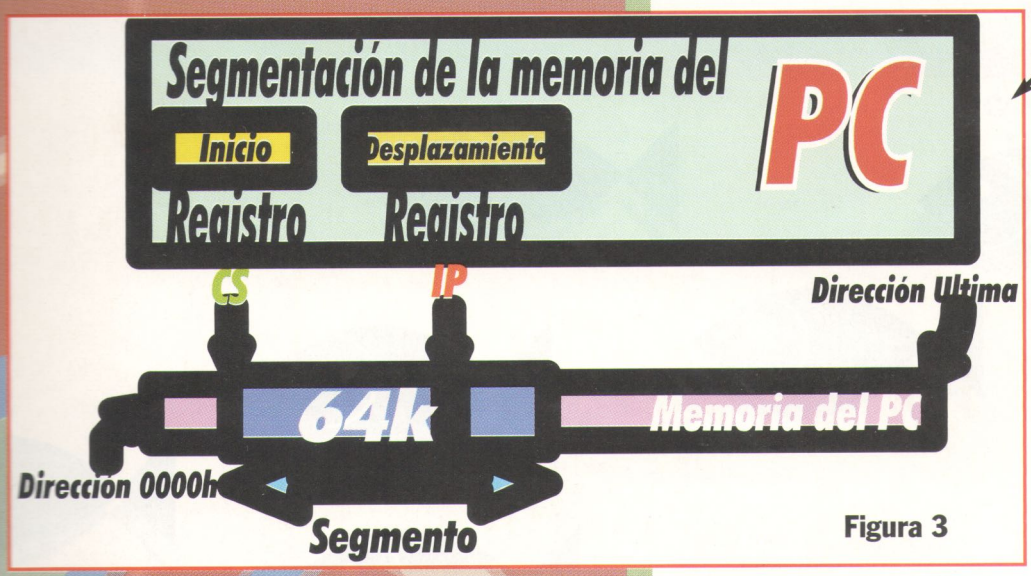


Figura 3

En la figura podemos observar como la memoria del PC se divide en secciones de 64 kilobytes (65536 bytes), que se denominan segmentos de memoria. El registro CS, en este caso, actúa como puntero de el inicio y el registro IP como el OFFSET o desplazamiento dentro del mismo.

La memoria es tratada por la CPU como si se tratase de un mundo enorme, donde él sólo puede ver una pequeña porción a la vez a la que se llama segmento.

la figura adjunta.

Como podéis observar, el microprocesador segmenta o particiona la memoria, de manera que siempre necesita un comienzo de segmento, el cual se puede encontrar en cualquier lugar de la memoria, y un desplazamiento dentro del mismo que indica a que dirección, dentro del segmento en cuestión, estamos haciendo referencia para cualquier tipo de operación.

La segmentación de la memoria es materia muy importante y su comprensión resulta relevante al máximo para los programadores de lenguaje ensamblador. Por ello, vamos a seguir dándoos pistas para que llegéis a una pronta comprensión de ella.

MAS SOBRE LOS SEGMENTOS

Los siguientes números también son direcciónes de memoria:

1278:34BA
2345:3123
1256:FFFF

Como podéis comprobar, no se añade la H detrás de estos números hexadecimales ya que la notación XXXX:XXXX siempre indica que las cifras están por defecto en ese formato.

En realidad, la forma en la que el microprocesador actúa si tuviese unas gafas con protecciones laterales que solo le permitiesen ver una ventana dentro de toda la memoria global del PC.

Aunque os hayáis perdido y no entendáis mucho el porqué de la segmentación de la memoria no os preocupéis. El próximo mes habrá más.

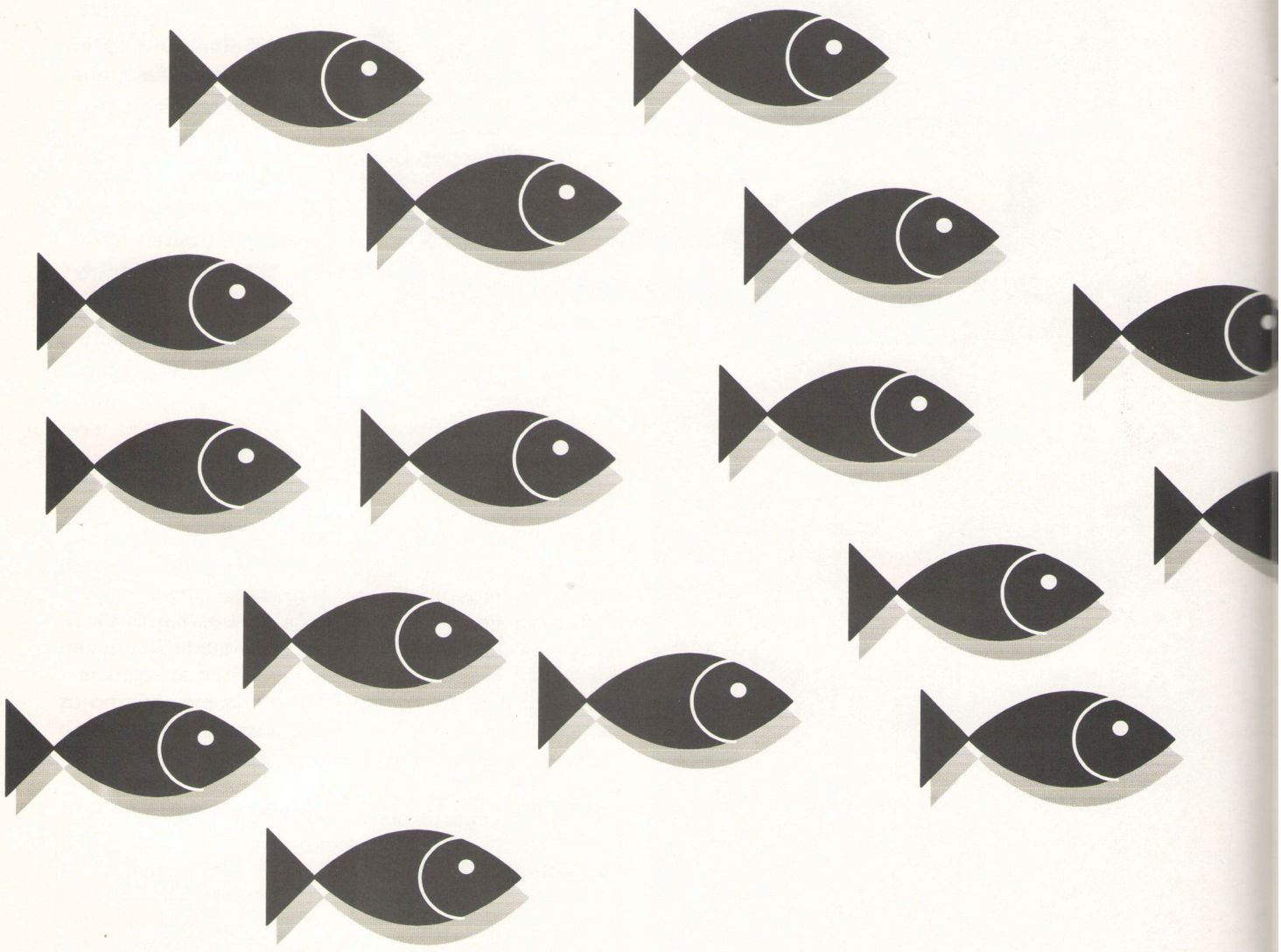
Tabla de conversión hexadecimal a decimal

Posición	Hexadecimal	Decimal
0	16^0	1
1	16^1	16
2	16^2	256
3	16^3	4096
4	16^4	65536
5	16^5	1048576
6	16^6	16777216
7	16^7	268435456

Ejemplo: 2A1F
2 x 4096= 8192....Posición 3
A x 256=2560.....Posición 2
1 x 16= 16.....Posición 1
F x 1= 15.....Posición 0
Total= 10783

Figura2, Cómo convertir HEX a DEC

No vaya en contra de todos...

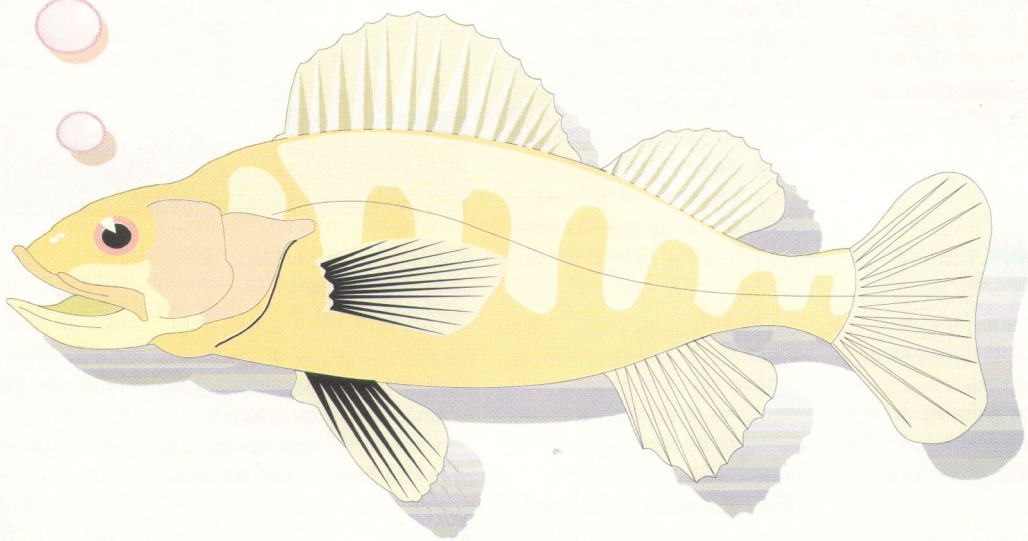


Asinfo

Asesoramiento y suministros informáticos

Estamos en el teléfono
y
FAX
956-813129

C/ María Auxiliadora 38
11520, Rota, (Cádiz)



En Asinfo encontrará su tienda amiga, su tienda de informática.
Con nosotros resolverá todos sus problemas con los ordenadores y los periféricos.
Con nosotros obtendrá los mejores precios, las mejores ofertas.
Pida gratuitamente nuestro catálogo de productos.

CARTA DEL LECTOR

Envidia cochina



Por:

Luis Gomez Asturias

Me miran como a un bicho raro. La edad que tienes y siempre dándole a la palanca esa, matando marcianos y bichos raros.

Los padres ya se sabe, nunca están contentos. Aunque seas buen estudiante y las notas te vayan bien, como es mi caso, siempre parecen sacarse algo de la manga para darte la vara. Y mira por donde el motivo de la queja es, en esta ocasión, mi afición a los juegos. Y me pregunto yo, ¿por qué los padres odian los ordenadores y los videojuegos? Esta conclusión no la saco de la manga. Mi amigo José Luis está en el mismo caso. Buen muchacho, estudioso y no muy callejero que digamos, pero mira por donde no le permiten estar más de tres horas seguidas con el ordenador. Excusas: dos mil diferentes. Desde "se te está quemando la vista" hasta "los ordenadores, estadísticamente, producen más casos de epilépsia que cualquier otra actividad" o "esto supone un gasto de luz considerable y tenemos que ahorrar". Yo creo que es envidia malsana. Sí, de verdad, cochina envidia. No quieren aprender, me refiero a los padres, y por ello lo pagan con nosotros. Se creen que el ordenador es algo complicado, difícil y austero. Todos los que trabajamos con ellos sabemos que, hombre, algo hay que estudiar al principio, pero estoy seguro de que si mi padre lo probase se daría cuenta que esto no es tan doloroso como una clase de latín en un colegio de curas.

¿A ver por qué no me puedo divertir con el ordenador?, si es que sois unos carrozas de mucho cuidado. Mi padre, a veces se me acerca por detrás cuando trabajo con mi PC y ya puedo estar programando en Dbase, que el siempre lo llama "comecocos". Y si a mi me gusta que me coman el coco, ¿por qué no voy a hacer lo que me gusta?

En fin, escribo esta carta porque sé, positivamente, que hay mucha gente que está en mi caso. Y que conste que en el pais del tuerto, el ciego es el rey. Quiero decir, mis padres dentro de lo que cabe y gracias a mis notas en el instituto, son permisivos conmigo y no me prohiben jugar o trabajar con el ordenador, pero imagino lo que tienen que pasar aquellos a los que se los esconden como castigo o simplemente para que no lo usen. ¡Vaya suplicio!

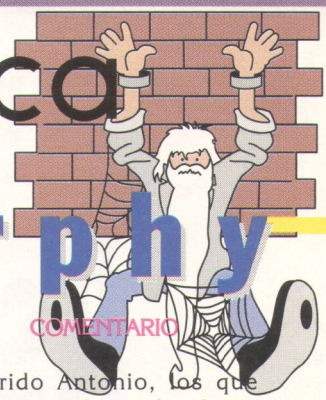
Si mi hijo quiere estudiar informática yo lo apoyaré en todo, porque considero que es una ciencia que, aparte de convertirse en una necesidad para muchos de los trabajos que se ofrecen hoy en día, nos ofrece la posibilidad de la autocreatividad. Como el que moldea el barro, esculpe, pinta, etc y en general todo aquel que cree o fabrique cosas, los ordenadores nos dan la oportunidad de hacer realidad muchas de nuestras ideas para mejorar nuestro negocio, llevar nuestros apuntes del colegio al día, aprender idiomas o algún instrumento musical.

¿Quién da más por tan poco dinero? Bueno, no gritéis, ya sé que en lo del dinero me he colado. Yo soy el primero que encuentra los artilujos varios que conectamos al PC muy caros en relación con el que gastaríamos si nos hubiese dado por otra afición diferente. Pero vuelvo a insistir que el gusto por la programación reporta satisfacciones de todo tipo.

Nada, ni siquiera cuando saco un sobresaliente, me ha llenado tanto de placer como el día, que tras varias horas de duro trabajo, conseguí hacer que el primer animación funcionase sin colgar el ordenador. O cuando la música que cree con la Soundblaster pude hacerla funcionar desde una aplicación creada por mí.

En resumidas cuentas, cuando uno se esfuerza en lo que le gusta, pronto recoge frutos. En un mensaje, que si me publicáis la carta, me gustaría hacer llegar a todos los que leen esta revista, a la que pienso va llegar muy alto por haberse acordado de todos nosotros, los que pasamos olímpicamente de leer estupideces sobre productos del mercado informático, muchas veces carísimos o de escaso o difícil uso, y queremos aprender algo que nos sirva el día de mañana. Me ofrezco, en mi humilde condición de programador novato a ayudaros en la medida de lo posible, si mis padres, claro está, me lo permiten.

La tintilla informática



La ley de Murphy

Por:

Miguel A. Lobato

Hola de nuevo amantes de la informática, nuestra sección se congratula por las cartas recibidas y os quiere animar para que continuéis enviándonos más cartas quejas.
¡Que mundo este! Es increíble las cosas que os pasan.

EXPOSICION DE LOS HECHOS

Antonio González es un lector habitual de Código Juego. Desde el principio ha estado con todos nosotros en secreto, pues se confiesa bastante tímido, y por fin ha decidido escribirnos para hacernos partícipes de un hecho muy singular y particularmente extraño.
Son ya diez o doce veces las que este fenómeno le viene sucediendo, al principio lo atribuía a la casuística o al azar o como el nos dice "tengo a un fantasma dentro de mi ordenador".
La primera vez que le pasó, estaba merendando un "bocata" de tortilla, cuando de repente, al querer grabar en disquete un archivo, la unidad de disco no atendía sus órdenes.
Pensó, "ya está, le he dado a una tecla errónea al tratar de compaginar mi delicioso bocadillo con mi ordenador."
Transcurrido unos segundos, volvió a intentarlo y ¡zas! la unidad seguía sin responder.
Comprobó el disquete y todo estaba en orden.
" Ya se ha fastidiado la unidad. ¿Por qué no se enciende la lucecita?"
Se levantó, fue al cuarto de baño y tras lavarse las manos regresó a la tarea y... ¡Eureka! la unidad respondía con total y absoluta normalidad.
"¡Uf! menos mal, ya estaba haciendo números para comprar una nueva o para repararla."
La segunda, al tratar de encender su ordenador este emitía un pitido sordo y molesto que dejaba la pantalla en blanco y a su unidad central no operativa. Apagó y encendió, y nuevamente los pitidos. Cinco veces lo intentó y a la quinta su ordenador arrancó como si jamás le hubiera ocurrido nada de nada.
"¡Este ordenador tiene algo! no se exactamente lo que es, pero algo tiene. Cuando tenga tiempo lo llevaré a que lo revisen."
La tercera dos días después, Antonio quería editar un fichero TXT para leerlo y bueno... podéis imaginaros, buscó en el subdirectorio donde estaba trabajando y su editor estaba allí, el archivo también pero no se podía acceder a el.
"¡Mañana sin falta llevo este trasto a que lo revisen!"
Al día siguiente, embaló su ordenador y se dirigió a la tienda de reparación.
"Por favor, revisen el ordenador algo debe tener mal apretado o no se, pero falla más que una escopeta de caña."
A los tres días volvió a recoger su máquina, el dependiente le dijo:
"Hemos revisado a fondo el ordenador, no le ocurre nada, quizás algún golpe, no obstante lo hemos limpiado y repasado."
Antonio se marchó a su casa contento pues no quisieron cobrarle ni una peseta, ¡Qué bien trabajan esta gente de INFO..!
A la semana de trabajar con su ordenador sin incidentes, nuevamente volvieronlos "Duendes" esta vez era el CD-ROM, pedía la imagen de un coche de sus clippart y le aparecía la de una playa desierta. Catorce veces repitió la operación y hasta la última no obtuvo la del coche. Una y otra vez extrajo el CD, lo limpió... nada de nada.
"¡Que pasa! otra vez se ha fastidiado. ¡Chorizos vendedores!"
Nuevamente lo llevó a la tienda, tan pesado se puso que le prestaron otro ordenador hasta que el suyo estuviera "reparado".
Esa misma tarde y tras varias horas sin percances, al tratar de ejecutar un juego, la unidad central no respondía. Al realizar un reset todo era ok.
"¡Qué he hecho yo para merecer esto!"
A la semana y media entregó el prestado y recogió su computer.
"Lo hemos mandado a la casa central, los técnicos nos comentan que tu ordenador está correcto y que tras revisarlo hasta la saciedad lo han cerrado tal cual."
Al irse a su casa, no paraba de darle vueltas a lo que en menos de un mes le había sucedido. Qué extraño.
Ocho veces más en tan sólo una semana le ocurrieron cosas similares, al final harto de desmontar el ordenata y de intentar buscarle una explicación, se decía y repetía a si mismo: "Debo tener un Duende, Fantasma..."

COMENTARIO

Querido Antonio, los que hemos pasado muchas horas delante de ordenadores, podríamos contarte casos parecidos a los tuyos, si bien es verdad, no con tanta frecuencia pero ocurren tal y como tu las cuentas.
Al ordenador no le ha pasado nada, funciona perfectamente.
Nosotros por tratar de buscarle una explicación lo atribuimos a la LEY DE MURPHY.
Quizás te sorprenda, pero este término cada vez esta más extendido en el lenguaje informático. Quién lo inventó no lo sabemos, pero viene al pelo cuando algo no tiene una explicación coherente.
Al buscar en el diccionario al famoso Murphy nos dice:

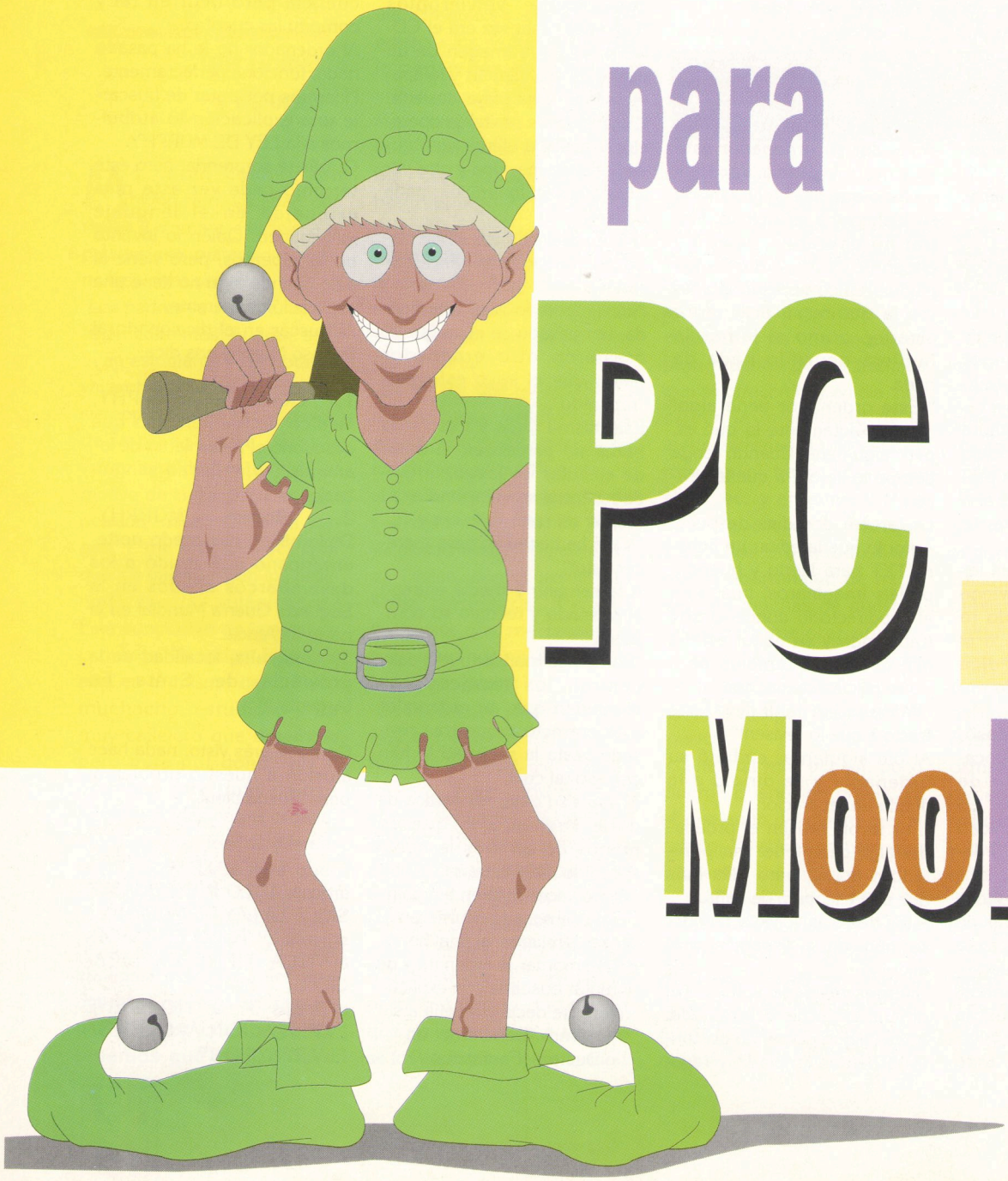
1.- Guillermo MURPHY, Médico norteamericano que descubrió el tratamiento de la anemia (enfermedad sanguínea).
2.- Robert MURPHY, Diplomático destacado norteamericano que ayudó a los desembarcos aliados en la Segunda Guerra Mundial en el norte de Africa.
3.- Geografía, localidad de la provincia de Santa Fe (Argentina).

Como habrás visto, nada hace alusión a nuestro problema, pero ¡Suena bien!

MORALEJA

articulo 1º LO INEXPLICABLE SIGUE LA LEY DE MURPHY.
articulo 2º TODO EN INFORMATICA TIENE EXPLICACION.
articulo 3º SI NO TIENE EXPLICACION APLIQUESE EL ARTICULO 1º.

Así se hace un juego para PC



4ª PARTE

Mookie

Por:

Ramón López

Han sido muchas las cartas y llamadas que hemos recibido en nuestra redacción referentes a esta sección. Ojalá para las demás obtuvieramos la misma acogida, ya que ello nos ayudaría a mejorar la revista para todos vosotros. Casi todos me pedís que vaya más despacio en mis explicaciones, y eso es lo que me he propuesto a partir de ahora. Sé que de lo que se trata es de enseñar y por ello nos vamos a tomar las cosas con más tranquilidad, os lo prometo.

Os mostramos como usar las tablas DACTABLE para el efecto de fundidos.

Oscureciendo la pantalla

Muchos programadores que siguen esta sección, ya sean aquellos que comienzan a estudiar el lenguaje ensamblador o por el contrario los que lo trabajan desde hace tiempo, me han hecho el mismo comentario desde que comencé esta sección práctica de Código Juego, hace ya cuatro números. Por lo visto voy rápido en mis explicaciones. Esto es lo maravilloso que tiene una revista; tener unos lectores tras tus espaldas que te van corrigiendo poco a poco, paso a paso. Por ello, de lo anterior tomo nota y acción.

Pensába comenzar en CJ4 el estudio de la utilización de la tarjeta Soundblaster, pero lo dejaremos para un poco más adelante, cuando el efecto de fundidos quede mejor explicado y sea comprensible para todos.

Esto es así ya que, por ejemplo, en el número anterior tuve que interrumpir la explicación sobre el efecto de fundidos para enseñar como se carga un archivo del disco a la memoria, materia ésta imprescidible para los conocimientos de cualquier buen programador. Por lo tanto y sin más dilación entremos en el estudio de como se realiza un *fundido* utilizando el lenguaje ensamblador.

DACTABLE

El mes pasado vimos que las variables BLOQUES, nos permiten almacenar toda la información que nos interesa a la hora de cargar un archivo, es decir, nombre, longuitud y posición de la memoria donde queremos introducir los datos. También estudiamos que nuestros archivos gráficos se encuentran en formato crudo: Paleta de colores (768 bytes) y cuerpo de la imagen en pixels en relación de un byte por píxel, que es la forma en que trabaja el modo 13h (19 decimal) 320x200, 256 colores. Hasta aquí todo bien.

A continuación expliqué que nuestros archivos gráficos los cargábamos en dos partes. Primero la paleta de colores, datos correspondientes a los 3 componentes rojo, verde y azul de cada uno de los 256 posibles que podemos utilizar simultáneamente en pantalla, y que constituyen la información colorista de los pixels de la pantalla en cuestión.



En la imagen podemos ver una representación de una paleta típica VGA de 256 colores. 16 filas x 16 columnas

¿Qué quiere esto decir?, bien sencillo. Cada pantalla que nosotros creemos en un programa gráfico para después introducirla en nuestro juego, ya sea como logotipo de presentación, cuadro de menús o escenario del propio juego, tiene una cantidad de colores determinado. Por ejemplo, si el escenario de nuestro juego es un bosque ténebre, es normal que los colores verdes y pardos sean los que más abunden en la paleta de 256 colores que tenemos a nuestra disposición. Pero suponed que nuestro personaje entra en un escenario distinto donde cambia el fondo, por ejemplo una playa soleada. En este caso los colores azulados, añiles y amarillos brillantes serán los más abundantes. De esto deducimos que cada fondo tiene una gama de colores diferentes, colores que nosotros podemos cambiar cada vez que queramos gracias al lenguaje ensamblador y su rapidez.

Lo que si esta claro es que si tenemos que cambiar los datos de la paleta de colores (256 colores x 3 componentes de color por cada uno -RBG- = 768 bytes) estos datos tienen que estar guardados en algún sitio. Y que mejor sitio que el disco.

La posibilidad de alternar los colores de las pantallas, accediendo a la los registros de la tarjeta VGA y cambiando los datos de DAC de vídeo, nos ofrece una gran libertad de acción a la hora de trabajar con la información gráfica, a la vez que dota a nuestros

programas de una vistosidad que es de agradecer, tanto por parte del programador como por el jugador.

Volviendo a nuestro programa, cada pantalla lleva en la cabecera del archivo estos famosos 768 bytes que nosotros podemos "toquetear" libremente. Esta es la razón de que cargemos los datos por separado.

Si habéis indagado el listado MAIN.ASM (por cierto, *main* significa PRINCIPAL, ya que será el módulo más importante de MOOKIE) os habréis dado cuenta que de los 64768 bytes que configuran cada pantalla gráfica, los primeros 768, es decir la paleta, se cargan en una dirección etiquetada como DACTABLE y el resto, 64000, que constituyen los pixels de la pantalla se cargan en la dirección A000:0000, pues bien, expliquemos el porqué.

La memoria de vídeo, situada a partir del segmento A000h en el modo gráfico 13h, es igual que la RAM del PC, es decir, son celdillas de memoria que son capaces de mantener la información como cualquier posición situada en la RAM convencional (los primeros 640k). Lo que hacemos es aprovechar este magnífico almacén, que al principio está inutilizado, para no tener que llenar de datos inútiles nuestra querida, escasa y preciada memoria de trabajo.

Todos sabemos que las pantallas de inicio, o sea las de presentación, no tienen más valor que el de impactar a la persona que comienza el juego, introduciéndola en lo que será el juego que viene a continuación, por tanto, una vez mostradas en pantalla pueden ser desechadas. Sería una incongruencia guardarlas en memoria de forma permanente para nada, ya que nos ocuparían mucho espacio, cosa que no nos podemos permitir, puesto que nuestro programa es pequeño ahora, pero crecerá considerablemente con el tiempo.

```
2A01:0D20  41 31 44 41 31 44 41 31-44 41 31 44 41 31 44 41   A1DA1DA1DA1DA1
2A01:0D30  31 44 41 31 44 41 31 44-41 31 44 41 31 44 41 31   1DA1DA1DA1DA1D
2A01:0D40  44 41 31 44 41 31 44 41-31 44 41 31 44 41 31 44   DA1DA1DA1DA1DA
2A01:0D50  41 31 44 41 31 44 41 31-44 41 31 44 41 31 44 41   A1DA1DA1DA1DA1
2A01:0D60  31 44 41 31 44 41 31 44-41 31 44 41 31 44 41 31   1DA1DA1DA1DA1D
2A01:0D70  44 41 31 44 41 31 44 41-31 44 41 31 44 41 31 44   DA1DA1DA1DA1DA
2A01:0D80  41 31 44 41 31 44 41 31-44 41 31 44 41 31 44 41   A1DA1DA1DA1DA1
2A01:0D90  31 44 41 31 44 41 31 44-41 31 44 41 31 44 41 31   1DA1DA1DA1DA1D
2A01:0DA0  44 41 32 44 41 32 44 41-32 44 41 32 44 41 32 44   DA2DA2DA2DA2DA
2A01:0DB0  41 32 44 41 32 44 41 32-44 41 32 44 41 32 44 41   A2DA2DA2DA2DA2
2A01:0DC0  32 44 41 32 44 41 32 44-41 32 44 41 32 44 41 32   2DA2DA2DA2DA2D
2A01:0DD0  44 41 32 44 41 32 44 41-32 44 41 32 44 41 32 44   DA2DA2DA2DA2DA
2A01:0DE0  41 32 44 41 32 44 41 32-44 41 32 44 41 32 44 41   A2DA2DA2DA2DA2
2A01:0DF0  32 44 41 32 44 41 32 44-41 32 44 41 32 44 41 32   2DA2DA2DA2DA2D
2A01:0E00  44 41 32 44 41 32 44 41-32 44 41 32 44 41 32 44   DA2DA2DA2DA2DA
2A01:0E10  41 32 44 41 32 44 41 32-44 41 32 44 41 32 44 41   A2DA2DA2DA2DA2
2A01:0E20  32 44 41 32 44 41 32 44-41 32 44 41 32 44 41 32   2DA2DA2DA2DA2D
2A01:0E30  44 41 32 44 41 32 44 41-32 44 41 32 44 41 32 44   DA2DA2DA2DA2DA
2A01:0E40  41 32 44 41 32 44 41 32-44 41 32 44 41 32 44 41   A2DA2DA2DA2DA2
2A01:0E50  32 44 41 32 44 41 32 44-41 32 44 41 32 44 41 32   2DA2DA2DA2DA2D
2A01:0E60  44 41 32 44 41 32 44 41-32 44 41 32 44 41 32 44   DA2DA2DA2DA2DA
2A01:0E70  41 32 44 41 32 44 41 32-44 41 32 44 41 32 44 41   A2DA2DA2DA2DA2
2A01:0E80  32 44 41 32 44 41 32 44-41 32 44 41 32 44 41 32   2DA2DA2DA2DA2D
2A01:0E90  44 41 32 44 41 32 44 41-32 44 41 32 44 41 32 44   DA2DA2DA2DA2DA
```

Forma en que DACTABLE se ve en memoria gracias a la directiva DUP

Pero hay un problema. El efecto de fundido ha de comenzar de negro a color, de color a negro, de negro a color, y así sucesivamente. En el momento que nosotros cargamos información desde el disco a la memoria situada en A000h, la paleta de colores que en ese momento está activa (por lo general la que funciona por defecto o de forma estándar en la VGA) hará que nuestros datos recién cargados aparezcan en pantalla de sopetón, con lo que el efecto de fundido nos sale mal desde un principio. Además, y para colmo, la paleta estándar VGA no tiene porque coincidir con la paleta de colores que hemos usado para crear nuestra pantalla, con lo que lo aquello que aparezca tendrá el cuerpo de pixels que nosotros le hemos dibujado pero los colores serán, con casi toda seguridad, distintos. Lo que visualizaríamos en ese momento sería un conglomerado de dibujos sin sentido y sin la personalidad que le atribuyen los colores correctos.

¿Cómo solucionamos esto? Si observáis el listado MAIN.ASM, en la página 17 de Código Juego 1 parte inferior de la columna situada a la izquierda, una vez que hemos cambiado el modo de vídeo al 13h, mediante las instrucciones:

```
MOV AX, 0013h
INT  10h
```

... rápidamente llamamos a una subrutina denominada APAGA, la cual está listada en la página 20 de este mismo número. Su función es la de apagar los registros de color del DAC de vídeo de la VGA. Es decir, se ponen a cero los componentes R, G y B de cada uno de los colores. Como sabemos, y si no lee los artículos sobre la tarjeta VGA de números anteriores, cuando los componentes RGB de cada color se van aproximando a 0, (te recuerdo que pueden valer entre 0 y 255) los colores se oscurecen. Cuando un color es R=0, G=0 y B=0 es color que obtenemos es el negro. Por lo tanto APAGA realiza esta operación de poner a cero los 3 componentes RGB de cada color. Gracias a este truco, cuando introduzcamos un byte-pixel en la memoria de vídeo (situada a partir de A000:0000) no aparecerá nada en la pantalla puesto que los 256 colores posibles de la paleta son todos NEGRO en ese momento. Gracias a la rutina CARGA, que vimos en el número anterior, podemos introducir los datos del cuerpo de la pantalla directamente en la posición donde aparecerán finalmente. La información estará allí, pero no se verá hasta que nosotros así lo queramos.

Bueno, pero si he puesto todos los colores a negro, ¿cómo voy a ver los colores reales? Ahí es donde entra el truco de fundido, ya que en DACTABLE hemos cargado los valores correctos de la paleta de colores de nuestra pantalla gráfica que también se ha introducido desde A000:0000 hasta A000:FA00, en total 64000 bytes.

Gracias a la paleta que hemos guardado, previamente, al comienzo de cada archivo, ahora podemos jugar a nuestro antojo.

LAS TRES TABLAS

Si os fijáis un momento en el segmento de datos de MAIN.ASM, en la página 21 de CJ1, veréis que justo al principio se reserva un espacio especial de la siguiente manera:

```
DACTABLE1 BYTE 256 DUP ("DA1")
DACTABLE2 BYTE 256 DUP ("DA2")
```

```
DACTABLE3 BYTE      256 DUP ("DA3")
```

En realidad son tres tablas de 768 bytes cada una. Lo que hacemos es guardar un sitio en blanco para el uso posterior en las rutinas que se encargan del efecto de los fundidos. Esto se logra gracias a la directiva, utilísima por cierto, DUP, con la cual duplicamos o rellenamos la memoria de los datos que se encuentran entre paréntesis. Es costumbre de los programadores, y la mía también, poner siempre un indicativo dentro de esos bloques "muertos" o vacios para que más tarde sea sencillo reconocer las partes del programa cuando miramos la memoria directamente. Por ejemplo, si usamos la directiva:

```
DACTABLE1 BYTE      768 DUP (0)
DACTABLE2 BYTE      768 DUP (0)
DACTABLE3 BYTE      768 DUP (0)
```

En la memoria reservada para DACTABLE1 se introducirán sólo ceros. Si utilizamos el comando D de DEBUG para ver la memoria no veremos sino una sucesión de ceros que nos ayudarán poco a ver en que parte comienza una tabla u otra. Ver figura en página anterior.

Esto es tan sólo un pequeño truco de programador que en nada afectará el buen funcionamiento de nuestro programa, sea cual sea el método que utilicemos, ya que en ambos casos se reservará el mismo espacio, que es lo que nos interesa. Pero se trata no sólo de programar sino de hacerlo bien y con corrección, por ello os iré dando estas pequeñas pistas todos los meses.

De las tres tablas, las dos primeras que vamos a analizar son DACTABLE1 y 3. DACTABLE2 la estudiaremos más adelante.

En la primera, se guardan los datos que entran desde el disco, es decir, la paleta de colores original de nuestra pantalla, que repito, también se encuentra ya cargada en memoria en la posición A000:0000 en adelante. En la etiquetada como 3, se van a guardar los datos de la primera y la segunda una vez que hayan sido tratadas por las rutinas del fundido. Por tanto es la número 3 desde la que tomaremos los datos que irán a los registros del DAC de vídeo de la VGA.

Bien, antes de cargar los datos gráficos, hemos de oscurecer la pantalla. No borrarla, que es diferente, ya que no tocamos ningún dato de la memoria de vídeo. Para ello, usamos APAGA. Esta rutina hace lo siguiente:

1) Pone el registro AX a cero con XOR AX,AX

2) Carga en el registro ES el contenido de DS, es decir, el segmento de datos

3) Carga en DI el valor del OFFSET, o comienzo, de DACTABLE3

4) Guarda el valor de DI para uso posterior mediante PUSH DI

6) Carga el registro contador CX con 384, justo la mitad de 768.

7) Rellena de ceros la tabla DACTABLE3 mediante REP STOSW, el cual almacena 2 bytes cada vez, por ello utilizamos 384 en el registro CX.

> Gracias a que podemos programar la tarjeta VGA y sus registros, conseguimos oscurecer la pantalla, con lo cual puede tener información pero no ver el contenido de la misma.

8) Una vez que la tabla está a cero se procede a cambiar los registros del DAC con esos valores, en este caso, se ponen todos a cero con lo que los 256 colores pasan a negro llamando a la subrutina DACVIDEO que es la que realmente realiza el trabajo de introducir los valores RGB en cada uno de los 256 registros de la VGA.

9) Se recupera el valor de AX que se guardó al principio mediante PUSH AX

Entonces salimos de la rutina para volver después de CALL APAGA en la página 17 de CJ1. Tras ello pasamos a la carga de los archivos como vimos el mes pasado, y que se encuentra englobada en la subrutina SHOW.

La situación en la que nos encontramos es la siguiente: Tenemos 768 bytes RGB, o de la paleta cargados en DACTABLE1 y 64000 bytes-píxel en A000:0000. Las tabla DACTABLE2 está vacia y la DACTABLE3 se ha quedado llena de ceros, pero no vemos nada en pantalla porque todos los colores están a negro.

El efecto de fundido consistirá en ir pasando progresivamente del negro al color original, es decir, convertir los datos de DACTABLE3 en los datos de DACTABLE1. Para ello guardaremos la información intermedia en DACTABLE2. ¿Comprendéis ahora el porqué del uso de las tres tablas de 768 bytes? Una es la inicial, otra la final y por último una que servirá como almacén intermedio. Cuando el contenido de DACTABLE2 sea igual que el contenido de DACTABLE1 habremos llegado al final del efecto, puesto que la información en pantalla habrá pasado de negro a el colorido final y original de la imagen.

Esto es, en resumidas cuentas como funciona este efecto tan vistoso y que en el próximo número aprenderemos a hacer, una vez que ya hemos aprendido su filosofía y otras muchas cosas más.

Sabemos apartar o guardar espacio en nuestros programas para zonas de memoria que van ha ser utilizadas como BUFFERS o sitios de trabajo donde una gran cantidad de variables han de almacenarse.

También sabemos cargar archivos y partes de archivos desde el disco a la memoria y como saber cuando ocurren errores. Sigue estudiando MAIN para el mes próximo.

Te ayudamos a escoger que juego realizar echando un vistazo al pasado

El pasado y presente del software lúdico.

Por:

Antonio Marchal Cortes



LAS PRIMERAS MAQUINAS

Aún recuerdo con nostalgia, la primera vez que ví un ordenador. Corría el año 1983, yo estaba en séptimo de E.G.B., y con unos amigos me cole en la clase de electrónica. Allí, conectado a una tele en blanco y negro había un ZX81. El profesor encargado nos dijo, con gran alegría por mi parte, que un padre de un alumno, el cual trabaja en la I.B.M., nos iba a dar clases de informática, y encima gratis. Esas clases me sirvieron de mucho, porque desde aquel día de lluvia, que nos metimos en esa clase, mi vida cambio por completo. Al poco tiempo me compré un Spectrum +, recién salido al mercado. A los pocos años, cuando por fin se rompió del todo, lo cambie por un +3, y después de muchas calamidades, conseguí comprarme un compatible PC. Los juegos que han pasado por mis ojos, se podrían contar por cientos, ya que todas las semanas me veía tres o cuatro. Además, no dejaba de analizar cualquier máquina recreativa que pasaba cerca de mi, ni tampoco ninguna revista sobre el tema de los videojuegos. De todas esas imágenes y recuerdos hablaré ahora.

Tic...Toc...Tic...Toc...Tic-Toc, ese era el ruido que hacia una de las primeras máquinas que hubo, el ping-pong. Aunque de ping-pong tenía poco, no había mesa, no había jugadores, las raquetas eran dos palotes, y la pelota ni tan siquiera era redonda. Pero como no había otra cosa causo furor, y es más como ya veremos, esta idea aunque sencilla, se repitió en otros juegos de gran prestigio. La máquina era sencilla, con dos mandos, uno para cada jugador, que parecían el volumen de cualquier cassete.

Manejabas tu raqueta, para evitar que la pelotita se saliera por tu lado de la pantalla, a la vez que intentabas que saliera por la del contrario.

Yo por aquel tiempo, ya había tenido contacto con otra máquina, porque mi cuñado, que hacia la mili en Andorra, me había traído una maquinita de "contrabando". Esta era de coches, y tenías que hacer la mayor puntuación posible, evitando chocarte con otros coches. Tu coche era una lucecita verde, y la de los demás una amarilla. Era güai, porque se podía jugar en la oscuridad, es decir, era mejor jugar en la oscuridad. Esto lo cuento, para demostrar el gran cambio que ha habido en poco más de diez años.



Pero sigamos con el tema.

Otra de las máquinas que apareció por los bares por aquel tiempo, junto con las primeras tragaperras, y el típico juego de la "formula-peseta", fue el Invaders. Esta máquina, para el tiempo que corría era algo sensacional, tenía bichitos, un mando en condiciones, y cantidad de sonidos, o más bien ruidos. Tu misión era acabar con la horda de invasores que poco a poco iba bajando por la pantalla, acercandose a tí, a la vez que te freía a disparos. Tú, para protegerte, tenías unas "casas", que se iban destruyendo con los disparos que recibía. De esta máquina, muchos tendrán gran recuerdo, e incluso hoy en día, todavía se sacan versiones remodeladas de la misma.

También salió por aquel tiempo, una muy parecida al ping-pong, del que he hablado antes. Pero esta vez en su versión de machaca-ladrillos. El juego era para sólo un jugador. Y su misión consistía en acabar con todos los ladrillos que había en la parte superior de la pantalla. Para ello tenías que mandar una pelotita, evitando que saliera por la parte de abajo de la pantalla utilizando una raqueta. Esta idea se uso posteriormente, dándole más calidad, lease "Arkanoid", pero esa es otra historia que ya contaremos.

Pero ahora pasemos a una de las más importantes. El Pacman, o comecocos, eso si que era flipante. Tenía gran cantidad de colores, no como el Invaders, lo que salían por los altavoces, se podía llamar música, e incluso más de uno

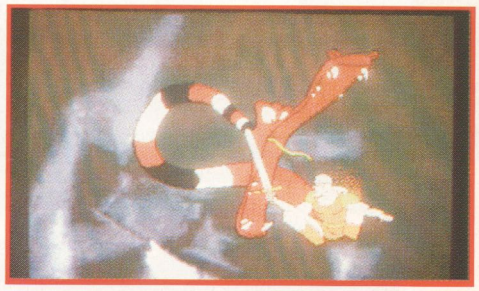
Wrath of demon AMIGA



PANG de AMIGA



Dragon´s Lair AMIGA



la tararearía más de una vez. Y aunque para los tiempos que corrían no se le podía pedir más, aún así, lo tenía, porque entre fase y fase salían graciosas escenas de introducción, con el comecocos de protagonista. En resumen una flipada, que pasará a la historia con letras de oro. ¿Qué...? ¡Qué no sabes de que iba el comecocos! No me lo puedo creer. Bueno te lo explicaré: en el comecocos tenías que dejar la pantalla vacía de cocos o puntos. Eras perseguido insistentemente por cuatro fantasmas, a los cuales si segundos antes te habías comido un super-coco o punto más grande, podías aniquilar. Cada nivel tenia una fruta distinta de bonus, y como ya he dicho, entre nivel y nivel aparecía 'Pacman y los fantasmas, haciendo escenas muy graciosas. Recuerdo una, en la que primero salía Pacman pequeñito perseguido por los fantasmas y un rato después aparecía gigantesco, apabullando a los pobrecitos fantasmas.

Después, llegó el gran boom de las máquinas, y empezaron a salir máquinas a troche y moche. Aparecieron las Moscas o Galaxy, el Pintor, Donkey Kong, Asteroids, el escalador loco o Crazy Climbers, Kung-fú, el minero, el Pengo, el Missile Defenders, Phoenix, Pole Position, Master kung-fu, Elevator Action, Hiper-olimpics, Spy Hunter, etc. Pero vamos a verlos más detenidamente.

Las Moscas o Galaxy ha sido un juego que ha tenido muchas versiones y aunque es bastante antiguo tiene una técnica y una adicción como pocos. Tu misión en este juego consiste en acabar con las hordas de moscas que te atacan desde la parte superior de la pantalla, cada vez que acabes con un batallón te dan una banderita.

El pintor o Brush, en este juego tú eras un rodillo que tenía que pintar toda la pantalla, por trozos, rodeándolos, mientras escapabas de tus enemigos. Cada nivel tenía un simpático bonus, tipo tragaperras, en el cual había cuatro caminos y elegías pulsando una tecla. Esta idea ha sido tomada hace poco, pero mejorándolo técnicamente, y agradándote la vista con bellas señoritas en paños menores.

El Donkey Kong ha tenido varias versiones, yo he visto cuatro, pero aquí , por ahora, hablaremos de la primera. En ésta fue donde apareció por primera vez el super conocido Mario Bros, al cual tu controlabas. Debías conseguir salvar a tu amada del terrible Donkey Kong, tenía cuatro fase distintas, y para defenderte, a veces, podías coger un martillo.

El Asteroid fue una de las primeras máquinas que uso gráficos lineales. En ésta, tu única meta era el sobrevivir, a una oleada de meteoritos que venían por todas partes. Tu nave podía girar y avanzar con una peligrosa inercia. Cuando disparabas a los meteoritos, se dividían en dos mas pequeños, lo que incrementaba el número de de ellos hasta cantidades demenciales.

El escalador loco o Crazy Climbers, fue una de las máquinas que a mi más me impresionó. Me tiré horas oyendo el "uandale", que la máquina soltaba cada vez que te parabas. En esta máquina tu tomabas la función de escalar edificios. Salía gente por las ventanas que tiraba absolutamente de todo y te impedía subir. También salían gorilas, pajarracos que te cagaban huevos y carteloncios que te podían machacar. Cuando llegabas arriba un helicóptero te esperaba. Destacaban sus sonidos, así como su adicción. Era un poco complicada, porque llevabas dos mandos.

Kung-fu fue una de las primeras maquinas que hablaban junto con Crazy Climbers. Cada vez que empezaba un combate, con dos karateka, el maestro de ceremonia decía "BEGIN", además de anunciar luego al ganador. Los que halláis visto la película de Contacto Sangriento habréis visto la máquina en donde Vall Damme reta a su amigo el barbas. Otra peculiaridad

Xout de AMIGA



Xout de AMIGA



que tenía esta máquina era que llevabas al muñeco con dos mandos, y si jugaban dos, uno contra uno, eran cuatro mandos, esto hacía que el numero de golpes que tenías para atacar a tu contrario era enorme.

El minero o Gilinga Gold, fue una máquina sencilla, pero que todavía es recordada por algunos con añoranza. Yo tenía un amigo que jugaba también a ésta, con cinco duros se tiraba horas, que incluso en algunos sitios donde la tenían, no le dejaban jugar. Es esta máquina tu eras un minero y debías recoger todo el oro que en ella había. Tenías carretillas en donde te podías montar, para evitar a los enemigos. Tambien tenía sonidos que imitaban voces.

Pengo fue la primera máquina de la super poderosa Sega. En esta tu eras un pingüino que debía juntar tres diamantes, empujándolos, para pasar de pantalla.

Como véis los comienzos ahora se recuerdan con nostalgia.

Y esto es todo por este mes. En Octubre continuaremos viendo más juegos y como estos han evolucionado a lo largo de la historia. Hasta el mes que viene.

Letras en movimiento

Aprende como mover el texto en pantalla

Scroll

Por:

Francisco Vinuesa

Hola de nuevo, aquí FRAN, vuestro colaborador de Valencia . Si ya habéis digerido el artículo de movimiento del mes pasado, os he preparado algo más.

No tenía muy claro que tema tratar este mes, pero varios lectores me han preguntado acerca de cómo se hace un SCROLL de texto como el que vieron en BYE mini-demo. La verdad es que les comenté que se trataba de un simple efecto que ni siquiera utiliza los registros de la VGA pero aún así, querían saber como se hacía, por lo que aquí tenéis el listado lo más limpio y comentado posible.

La idea es el hacer que el texto de la impresión de mueva suavemente de izquierda a derecha. Ello basa su lógica en que los españoles y algunos otros leemos en esta dirección, (la versión japonesa de CJ presentará este efecto de arriba para abajo, claro).

Para hacerlo, tan sólo debemos imprimir el número de caracteres que caben a la vez en una línea de pantalla cada vez un poco más a la izquierda y pasando de carácter cada X veces que es el ancho del caracter y teniendo en cuenta que el de más a la izquierda se saldrá por ese lado y no se tiene que imprimir entero algunas veces y el de más a la derecha también hay que tratarlo de manera particular .... además tenemos que hacerlo muy rápidamente para evitar efectos cortina y otras incongruencias ...

Llegado a este punto uno se da cuenta de que hay dos maneras de llevar esto a cabo : La complicada sería teniendo todos esos factores en cuenta. Se podría hacer, pero como siempre queremos realizar las cosas de la manera más sencilla posible la haremos de otra manera (seguro que hay muchas maneras de llevar a cabo el efecto pero por sencillez, dado que no queremos usar el Modo X todavía, y para que el listado sea breve pero que funcione, vamos a hacerlo usando un bloque de memoria como pantalla "Virtual"). Que ¿qué es una pantalla virtual ? , seguro que los ingleses tienen algún nombre sonoro para ello, pero yo llamo así a utilizar una zona de memoria libre, como es en este caso del segmento de datos, para escribir la línea de texto completa y luego transferir esa línea rapidamente a la pantalla con instrucciones MOVSD , con lo que conseguimos, esperando el RETRAZO de la pantalla, evitar parpadéos y demás.

Así, al imprimirlo en memoria, podemos hacer que nuestra línea virtual sea de más de 320 pixels con lo que evitamos el problema de el caracter que sale (izq.) y el que entra (der.) y además nos podemos ahorrar el imprimir cada carácter de la línea 12 veces con un pequeño truco que es imprimirlo la primera vez y las otras 11 sólo hay que cambiar el inicio de la línea en memoria una dirección a la derecha. Dicho esto, no queda casi nada que explicar, el listado ya va ampliamente comentado por lo que es mejor remitirse a él para una mejor comprensión. Otra cosa, para facilitar el entendimiento del mismo, he separado en tres archivos el programa completo, siendo el principal el llamado SCROLL.ASM. Todo se debe ensamblar y linkar usando MAKE de Borland para lo que incluyo la MAKEFILE. Los otros dos son, digamos misceláneos, y una vez empieza el SCROLL, lo principal está como he dicho en SCROLL.ASM

FRAN94

```
SCROLL.EXE: SCROLL.OBJ TEXTO.OBJ CARGADOR.OBJ

        TLINK  /s /3 /X SCROLL TEXTO CARGADOR /MAP /V;

SCROLL.OBJ:       SCROLL.ASM
        TASM    /ZI /Q /M2 SCROLL;
TEXTO.OBJ:        TEXTO.ASM
        TASM    /ZI /Q /M2 TEXTO;
CARGADOR.OBJ:     CARGADOR.ASM
        TASM    /ZI /Q /M2 CARGADOR;
```

Makefile del programa SCROLL.EXE



Juego de caracteres utilizado en el efecto

En esta sección es donde publicamos vuestros artículos de gran tamaño. Ya lo sabes, ganarte un buen dinerito y ver tu artículo publicado en Código Juego es tan simple como que nos envíes tu colaboración en un disquete a la dirección:

CODIGO JUEGO
APDO. 114
11520 ROTA CADIZ




```
;CARGADOR.ASM Contiene el procedimiento de carga del fichero con los datos y
;unos punteros al segmento de datos para meterlos y operar con
;ellos
.MODEL  SMALL
        ASSUME  CS:_TEXT, DS:_DATA
.CODE
        PUBLIC  CARGA_SCR
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; Carga la pantalla que contiene los gráficos en el BUFFER sito en DS:DX
CARGA_SCR     PROC  NEAR

        MOV   AX,3D00h                  ;Abrir fichero
        MOV   DX,OFFSET [FICHERO]       ;DS:DX nombre de archivo
        INT   21h
        JNC   S_BIEN                    ;Fichero abierto
        MOV   DX,OFFSET [ERR1]          ;No se pudo abrir, error
S_ERR:  MOV   AH,9                      ;Muestra error y sale al DOS
        INT   21h
        MOV   AH,4Ch
        INT   21h
S_BIEN: MOV   BX,AX
        MOV   AH,3Fh                    ;Cargar fichero
        MOV   CX,11144                  ;Longitud
        MOV   DX,OFFSET BUFFER          ; en buffer tras programa
        INT   21h
        JNC   S_CERR                    ;Si bien, seguir

        MOV   DX,OFFSET [ERR2]          ; si no, salir con error,
        JMP   SHORT  S_ERR
S_CERR: MOV   AH,3Eh                    ; cerrar fichero
        INT   21h
        RET
CARGA_SCR     ENDP
DATA
FICHERO         DB    "CHARSET",0
ERR1            DB    "No se puede abrir fichero CHARSET $"
ERR2            DB    "Fallo de carga $"
PUBLIC          BUFFER, L_BUFFER

BUFFER = $                              ;Tras los datos,
     DB  11200 DUP (0)                  ; reserva espacio para gráficos
L_BUFFER = $                            ;Buffer para la línea de scroll
DATA   ENDS

END
```

```
Comment &
Para ensamblar utilizar MAKE de Borland ( TASM / TLINK 2+ ) (c) Borland Int.
Ejecutar con SCROLL
SCROLL Realiza el scroll de un texto en modo gráfico 320*200 256 colores

ARCHIVOS ASOCIADOS:
SCROLL.ASM      Bloque principal, es esto.
TEXTO.ASM       Contiene el texto a imprimir y el conversor de códigos
CARGADOR.ASM    Carga la paleta y el juego de caracteres
CHARSET         Contiene los gráficos de los caracteres y la paleta
MAKEFILE        Archivo necesario para MAKE
SCROLL.EXE      Programa listo para funcionar (sin Debug Info)
&
        DOSSEG
        .MODEL  SMALL
        .386
        ASSUME  CS:_TEXT, DS:_DATA
        .CODE
        EXTRN   CARGA_SCR:PROC, CONVERT:PROC      ;Procedimientos externos

;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::;
 Entrada del programa
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
PRG:    CLI                     ;Desconectar algunas interrupciones
        MOV     AX,PILA
        MOV     SS,AX           ;Poner el segmento de pila
        MOV     SP,400
        MOV     AX,DGROUP
        MOV     DS,AX           ;Ajustar DS
        CALL    CARGA_SCR       ;1º Cargamos la paleta y los gráficos de
                                ;las letras
        MOV     SI,TEXT_D       ;=OFFSET TEXT2 lugar donde empieza el
                                ;texto
        CALL    CONVERT         ;Como no hemos definido más que 54
                                ;caracteres, hay que hacer una conversión
                                ;de los ASCII
        MOV     AX,13h
        INT     10h             ;Modo gráfico
        MOV     CL,2
        CALL    PALETA          ;Pone la paleta original
        MOV     AX,0A000h
        MOV     ES,AX           ;Segmento de video

MAIN:   CALL    TEXT_SCROLL     ;Llama al corazón del programa

        MOV     AL,[T_ESC]      ;Comprueba si se pulso ESCAPE
        OR      AL,AL
        JNZ     NO_ESC          ;Si no, sigue su curso normal

        MOV     CL,[T_FADE]     ;Si llega aquí, realizar el fundido
                                ;(FADE_OUT)
        PUSH    CX              ; suave, para ello se coge un contador
        SHR     CL,4            ; que se desplaza para que a PALETA le
                                ;llegue un
        CALL    PALETA          ; valor entre 2 y 8 (necesario para
                                ;fundido)
        POP     CX
        INC     CL
        MOV     [T_FADE],CL
        CMP     CL,120          ;Al llegar a 120, se hizo la oscuridad ...
        JNE     NO_ESC
        JMP     SALIR           ; y sale

NO_ESC: CALL    RETRAZO         ;Esperar la señal de sincronía

        IN      AL,60h          ;¿ Se pulsó ESCAPE ?
        CMP     AL,1
        JNE     SEGUIR          ;No, seguir
        XOR     AL,AL           ;Si, activar la señal para fundido
        MOV     [T_ESC],AL
SEGUIR: JMP     MAIN
SALIR:  STI                     ;Activa interrupciones (bueno)
        MOV     AX,3            ;Modo 80*25
        INT     10h
        MOV     AX,4C00h        ;Salida al DOS
        INT     21h
;:::::::::::::::::::::::::::::::SINCRO:::::::::::::::::::::::::::::::
RETRAZO  PROC
```

Scroll.asm

```
        MOV   DX,03DAh          ;Esperar a la señal de sincronía
SINC:   IN    AL,DX
        TEST  AL,08
        JNE   SINC
SIN:    IN    AL,DX
        TEST  AL,8
        JE    SIN
        RET
RETRAZO ENDP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Pone la paleta y opcionalmente, si se le envia un valor en CL diferente
; de 2, se hace un efecto de fundido
; RECIBE    CL numero entre 2 y 8. 2 pone paleta normal, más la oscurece
PALETA  PROC

        CLD                     ;Quiero que LODSB incremente el SI
        MOV   SI,OFFSET BUFFER+10376 ;La paleta está tras los gráficos
        MOV   CH,0              ;256 registros de paleta
        XOR   BL,BL             ;Primer registro de paleta
B_OUT:  MOV   DX,03C8h          ;Indicar el nº de registro
        MOV   AL,BL
        INC   BL
        OUT   DX,AL
        INC   DX                ;DX=3C9h, recibe los componentes

        LODSB
        SHR   AL,cl
        OUT   DX,AL             ;Primer componente del color
        LODSB
        SHR   AL,cl
        OUT   DX,AL             ;Segundo
        LODSB
        SHR   AL,cl
        OUT   DX,AL             ;Tercero
        DEC   CH
        JNZ   B_OUT             ;Repetir 256 veces
        RET
PALETA  ENDP
.DATA
        EXTRN   BUFFER:WORD             ;Valores externos
        EXTRN   TEXT2:BYTE, TEXT_D:WORD, l_buffer:WORD

T_ESC   DB      1           ;Byte usado para saber si se pulsó ESC
T_FADE  DB      32          ;Primer valor del contador para el fundido
TEXT_Y  DW      120         ;Linea en donde se pone el texto
SCR_C   DB      2           ;Contador inicial a 2 para forzar la impresión de
                            ; la línea la primera vez
.CODE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Procedimiento que mira si debe imprimir la linea en memoria y la saca
; en pantalla

TEXT_SCROLL     PROC
        XOR     AX,AX
        MOV     AL,[SCR_C]          ;Contador de suma para 1er carac
                                    ;ter
        DEC     AL                  ;Contador - 1
        DEC     AL                  ; contador - 2 para más velocidad
        MOV     [SCR_C],AL
        JNZ     SIGUE               ;Si contador es 0,
        MOV     AX,[TEXT_D]         ; incrementar el puntero del
                                    ;texto,
        INC     AX
        MOV     [TEXT_D],AX
        MOV     SI,AX
        CALL    PINTAR              ; poner linea de texto en memoria
        MOV     AX,14               ;Repone contador a 14
        MOV     [SCR_C],AL
SIGUE:  CALL    MEM_TO_SCR          ;Saca texto de memoria a pantalla
        RET
TEXT_SCROLL     ENDP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Escribe una línea de texto de 25 caracteres en la zona de memoria
;l_buffer
; dejando un caracter vacio delante para el efecto de scroll
;RECIBE          SI = Offset del texto de la linea

PINTAR  PROC
        LEA     DI,l_buffer     ;Apunta a la zona de memoria reservada
                                ;para imprimir
        ADD     DI,14           ;14 bytes a cero para efecto de scroll
        PUSH    ES
        PUSH    DS
        POP     ES              ;Hacer ES = DS para STOSD
        MOV     CX,25           ;Nº de caracteres
SC2_25: LODSB                   ;Carga caracter
        CMP     AL,255          ;Comprueba si fin de texto
        JNE     SIGU2
        MOV     AX,OFFSET TEXT2 ;Pone texto al inicio si alcanzó final
        MOV     [TEXT_D],AX
        MOV     SI,AX           ;Actualiza el SI
        JMP     SC2_25          ; y repite la última operación
SIGU2:  PUSH    SI              ;Apunta próximo caracter
        PUSH    DI              ;Guarda posición en pantalla
        LEA     SI,BUFFER       ;Donde empiezan los gráficos del SET
        MOV     AH,0            ;Calcula inicio del caracter
        MOV     DX,12*16        ;Ancho de caracter * Alto
        MUL     DX
        ADD     SI,AX           ;Apunta SI al primer byte del caracter
        MOV     DX,355          ;Una linea (+1 caracter vacio delante +1
                                ;final
                                ; - 1 para doblar los caracteres(cursiva)
        MOV     BL,16           ;ALTO
ES2_12: PUSH    DI
        LODSD                   ;Transfiere 12 bytes, ancho de cada
                                ;caracter
        STOSD
        LODSD
        STOSD
        LODSD
        STOSD
        XOR     AX,AX           ;Dejar espacio vacio entre caracteres
        STOSW
        POP     DI
        ADD     DI,DX           ;Pasa a la línea inferior
        DEC     BL
        JNZ     ES2_12          ;¿Todo el caracter?
        POP     DI
        ADD     DI,14           ;Pos. inicial + 14 del caracter para
                                ;separación
        POP     SI
        LOOP    SC2_25          ;¿Todos los caracteres?
        POP     ES
        RET
PINTAR  ENDP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Pasa la linea de texto escrita en memoria al buffer de video
;RECIBE          AX      Valor de resta para efecto scroll,
;AQUI ESTA EL MEOLLO DE LA CUESTION
MEM_TO_SCR      PROC
        PUSH    AX
        MOV     AX,[TEXT_Y]             ;Calcula la linea en pantalla
        MOV     DX,320
        MUL     DX
        MOV     DI,AX
        POP     AX
        MOV     SI,OFFSET l_buffer+28   ;Inicio de la línea de caracteres
                                          ; en memoria + 28,
        SUB     SI,AX                   ; restamos el contador de SCROLL
                                          ;SI = inicio
                                          ;adecuado para scroll
        MOV     BX,16                   ;Alto de la linea

l_alto: MOV     CX,320/4                ;Ancho/4, usamos MOVSD
        REP     MOVSD                   ;Pasa 320 pixels a la pantalla
        ADD     SI,36                   ;Pasar el caracter vacio inicial y
                                          ;final
        DEC     BX                      ; y uno extra por la cursiva 12*3
        JNZ     l_alto                  ; y pasa a línea siguiente si
                                          ;procede
        RET
MEM_TO_SCR      ENDP
        ENDS



PILA    SEGMENT USE16

DB  400 DUP (?)

PILA    ENDS

        END     PRG
```

Por:

Redacción de CJ

La bibliografía sobre programación y sobre todos aquellos temas relacionados con esta es enormemente abundante. Os ayudamos en esta sección a escojer aquellos manuales que tienen más calidad

El libro del mes

Como todos sabemos la familia de microprocesadores de la compañía norteamericana Intel Corporation ha sido la que ha dominado el hardware del PC desde que éste nació hace ya más de una década. Desde el obsoleto y carismático 8086 hasta el Pentium de nuestros días, muchas cosas han cambiado tanto para los microprocesadores de esta amplia familia como para los directos consumidores de los mismos, es decir, tú y yo.

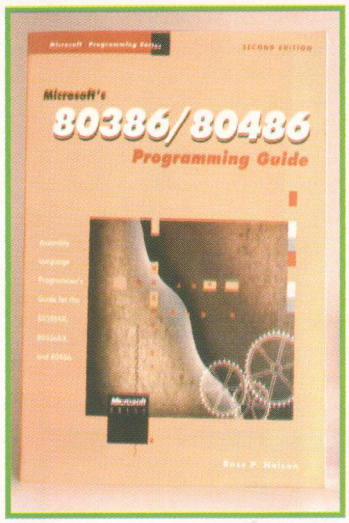
Han surgido competidores, eso sí, pero ninguno ha logrado hacer mella en la calidad de ellos, que siguen su paso firme hacia un futuro prometedor, incluso cuando suenan los ecos de próximas supermáquinas, léase PowerPC.

Nuestros queridos 80x86, como siempre se les ha llamado, constituyen la pieza vital y el hilo conductor que ha trazado los cambios de la informática moderna, la informática de casa, la que todos conocemos y amamos.

El libro que hoy comentamos, editado por Microsoft Press, está en inglés y ha sido escrito por Ross P. Nelson. No es ello una gran traba, me refiero a lo del idioma. La estructura técnica de la obra no nos impide entenderla en su gran mayoría, incluso cuando el idioma anglosajón no sea el que aparezca con mejor nota en nuestras evaluaciones escolares. Lo que si os diré es que, aparte de venir el manual avalado por una gran editorial como es la que mencioné antes, se trata de un libro imprescindible a la hora de dar el salto hacia la programación en modo protegido de los microprocesadores motivo de la obra.

Pero analicemos paulatinamente que llevan dentro estas 476 páginas.

En primer lugar, el autor es consciente de que lo que es preponderante sería explicar como ha evolucionado la familia 80x86 desde sus inicios. El 8080, 8086, 80287, 80286, 80287 son desglosados en estas páginas iniciales, en las que se nos hace ver las diferentes mejoras que han sido incluidas en estos procesadores de forma escalonada. Por supuesto, al final de esta evocación del pasado se desemboca en los procesadores de la gama alta, los más comprados y usados en la actualidad; el 386 y el 486.



Se nos antoja de gran importancia el estudio de estas primeras páginas, ya que nos ayuda en gran medida a entender el porqué de la necesidad de avanzar, de mejorar y de agradecer el hecho de que gracias a el mejor manejo de la memoria y rapidez de estos nuevas CPU´s, nuestros programas actuales son, cada vez, más perfectos.

Si queremos información técnica y críptica sólo tenemos que referirnos a los manuales que la propia compañía Intel edita. En ellos, y a nivel de analistas de hardware y personas muy profesionales y preparadas, encontramos datos que nos pueden ayudar en gran medida a la hora de profundizar, digamos "exageradamente" en los entresijos del microprocesador. Sin embargo, en el libro que hoy comento, se halla la información que de verdad nos puede interesar a nosotros, los programadores de videojuegos. Y es que cada día más, el modo real del procesador se queda más corto a la hora de hacer los maravillosos juegos que salen al mercado.

Por lo tanto, todos aquellos programadores interesados en dar el salto, este puede ser un primer escalón.

Tras es el desglose de la familia Intel, el autor nos explica cual es la estructura interna de la CPU. La unidad de interface de BUS (BIU), la unidad de prefetch, la unidad de descodificación de instrucciones, la unidad de ejecución, la unidad de segmentación y la de paginación son estudiadas a fondo para que aprendamos cual es la interrelación de los diferentes componentes que permiten que las instrucciones de nuestros programas sean llevadas a cabo.

Tras ello, se explica cual el método de segmentación de memoria. Es decir, como la CPU trabaja con su almacén de datos e instrucciones. Entonces se pasa al juego básico de instrucciones o mnemónicos con los que contamos para trabajar.

Después se entra en la materia interesante; el mecanismo de protección de memoria, gracias al que varios programas coexisten funcionando en la memoria sin que los datos o malas instrucciones de uno puedan dañar la estructura ejecutiva de sus programas compañeros.

Para finalizar la obra se explican todos los métodos de trabajo en los que estos procesadores pueden funcionar. Gracias a ello, procesadores más avanzados como el 386 y 486 pueden ejecutar software que ha sido escrito para CPU´s más antiguas. Los modos real, protegido y virtual son estudiados de forma que el aprendiz comprenda cuales son los avances y las restricciones de cada uno de ellos.

Como colofón del libro se publican una serie de tablas de referencia a la que podemos acudir en caso de que algo se nos resista durante el estudio de los capítulos anteriores.

Ficha técnica

Título: 80386/80486 Programming Guide

Editorial: Microsoft Press

Pag: 476

Autor: Ross P. Nelson

NUESTRA VALORACION

Se trata de una obra copiosa en datos, especialmente su sección de referencia que ocupa más de doscientas páginas. El único inconveniente que le vemos es el idioma, pero todos los que programamos sabemos que esta es una traba a la que nos tenemos que acostumbrar, especialmente cuando nos damos cuenta que no tenemos, por ello, que enfrentarnos a alguna que otra mala traducción que en más de una ocasión, nos ha hecho dar un libro por inútil.

Os lo recomendamos.

El juego del mes

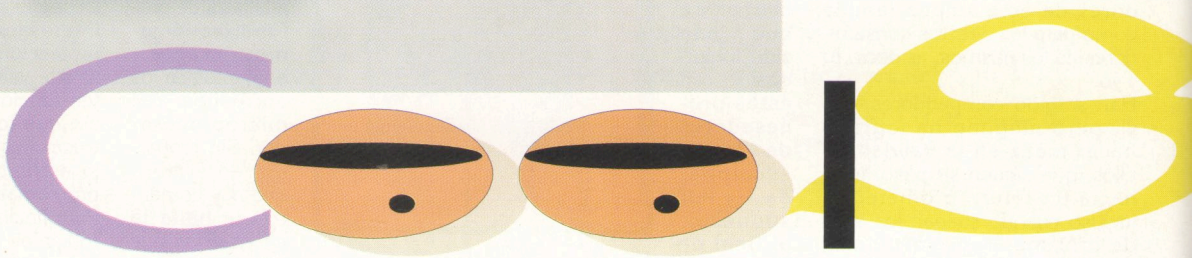
CODIGO JUEGO

Por:

Raul Reales

Nuestro colaborador escoge cada mes un juego que este en boca de todos. Nos intersa, especialmente, que el programador vea las diferentes formas en que se puede desarrollar un juego





LAS PLATAFORMAS

Desde el principio, la programación de juegos basada en plataformas, ha estado siempre presente. Y si esto entra en duda, tan sólo tenéis que retroceder en el tiempo y recordar ese mítico Jet-Pac que residía en una antigua cassette y se cargaba en el viejo ZX-Spectrum.

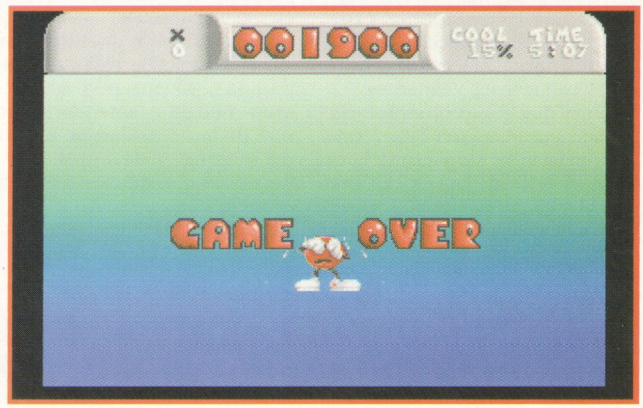
La filosofía a seguir en este tipo de juegos es, casi siempre, muy simple. Nuestro personaje debe recorrer un mapeado, por donde recoger unos items para llegar así a completar el nivel. Esto no ocurre siempre, ya que existen en el mercado juegos de plataformas en los que la complejidad puede llegar a niveles increíbles.

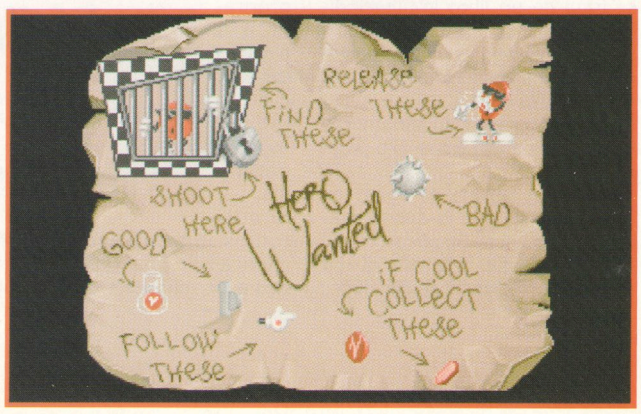
Una vez que hayas recogido los items, debe de conseguir encontrar la salida del nivel para pasar al siguiente. Esa es la mecánica de casi todos los juegos de plataforma. Unas veces más complicados, otras veces menos, pero con el mismo desarrollo.

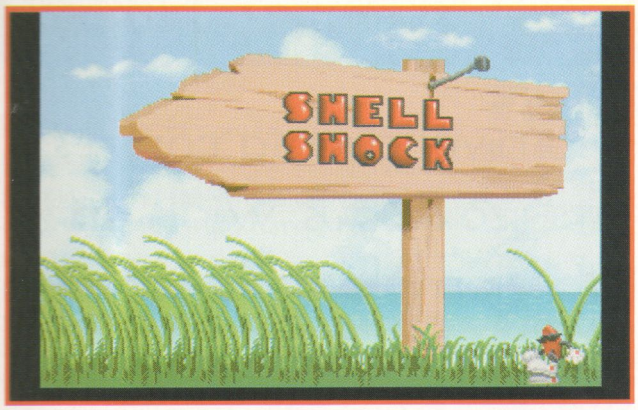
EL PROTAGONISTA

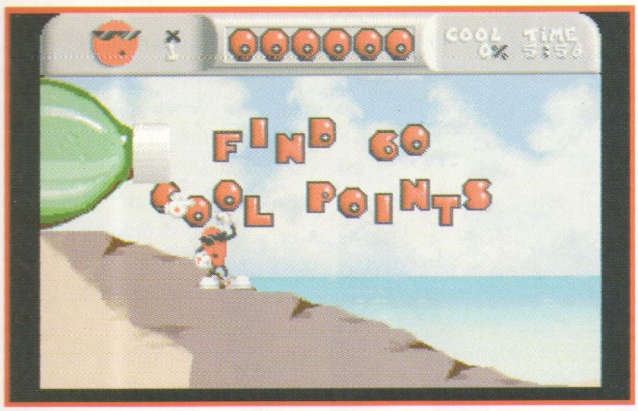
Este simpático personaje, seguramente, no os sonará de nada, aunque ya ha aparecido por segunda vez en un juego PC. En su primera aparición, hace algunos años, figuraba en un curioso juego de tablero donde debías conseguir cambiar de color las fichas de tu oponente. Este podía ser otro jugador -permitía jugar hasta cuatro- o el mismo ordenador. Además de lo que era el juego en sí -muy adictivo-, poseía unas animaciones que por esa época, y con una tarjeta VGA, te dejaban gratamente sorprendido.

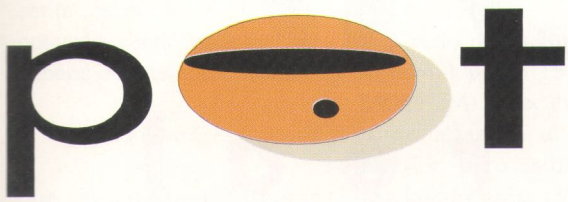
En esta segunda "entrega", nuestro amigo debe liberar a















otros compañeros que han sido atrapados. Esta vez se ha trasladado a las plataformas, para demostrarnos lo que sabe hacer.

EL JUEGO

Cool Spot esconde un gran despliegue técnico realizado por el equipo Virgin. Nada mas comenzar, empieza a sonar una musiquilla marchosa y pegadiza, que te introduce en la presentación. Ahí está él, Cool Spot en su botella, camino de rescatar a sus amigos.

El menú principal es muy simple y se divide en tres partes: Start Game, Options y Exit to Dos. La primera opción te sumerge de lleno en la aventura, que pasaremos a comentar más tarde. La segunda te permite configurar el sonido, los controles y el nivel de dificultad. La última... "no comments".

Una vez comenzamos, se nos muestra un papel como ayuda -muestra lo beneficioso y lo perjudicial-. A los pocos segundos, o la pulsación de una tecla, una nueva escena nos hace ver la zona en que nos encontramos. Al terminar esta, aparecemos en el comienzo del nivel con un objetivo: conseguir los items que nos especifican en pantalla. Comienza lo complicado. Tras esto te esperan 8 niveles y 7 niveles extra de bonos.

LAS HABILIDADES DE COOL SPOT

Nuestro punto rojo, con gafas y marchoso, posee una agilidad excepcional en todos los sentidos: puede realizar grandes saltos, piruetas, giros, etc. Aunque al principio pueda parecer un poco incómodo de manejar para los que no están acostumbrados a este tipo de juegos, luego será muy fácil de controlar. También, Cool Spot podrá defenderse mediante una especie de bolas puntiagudas brillantes, ilimitadas, y que podrá lanzar en todas direcciones y a la velocidad que quiera -eso ya es asunto de la rapidez de tus pulsaciones-.

Un dato curioso es su increíble habilidad para no estar nunca parado. Si se cae desde una gran altura, nada más levantarse se sacude el polvo, acción que realiza también al perder una vida y comenzar de nuevo. Si por cualquier motivo, lo dejamos quieto, sin hacer nada, comenzará a jugar al yoyo, limpiarse las gafas, bostezar... En fin, toda una maravilla en cuanto a lo que se refiere a la animación.

Como los programadores no sólo han pensado únicamente en el personaje principal, también han dotado a los enemigos de unas soberbias animaciones -en especial los cangrejos-. Desde luego, merece la pena tener el juego sólo para observar lo bien que han desarrollado las rutinas de movimiento para hacernos ver su calidad.

LA PROGRAMACIÓN

Lo que más destaca con respecto a las rutinas, es la suavidad del scroll -una envidia que tantas veces dejan de lado más de una vez este tipo de juegos. Caben mencionar los diferentes scrolls que aparecen a la vez, dando una sensación de planos, causando una agradable sensación a la vista. Por lo general, la rutina del agua -comienzo del programa- y otras de menos interés están bastante bien hechas. En general, ha sido un programa que no ha tenido ningún tipo de descuido en lo que respecta a realización. Nunca olvidemos que la mayor parte, sino todo el programa ha sido escrito en el lenguaje favorito de Código Juego: ensamblador.

VALORACIÓN

Aunque no se puede hablar de algo brillante, si se puede dar el calificativo de muy bueno, algo que realmente muestra las capacidades del PC a la hora de hacer las cosas bien. Virgin ha demostrado que las plataformas no deben quedar sólo para las consolas.

Este lenguaje estructurado de alto nivel cuenta con más adeptos cada día. Creado por el suizo Niklaus Wirth en la década de los setenta, es un claro ejemplo de la sencillez y, al mismo tiempo, la potencia.

2ª PARTE

Lenguaje PASCAL

El tipo real

Dentro de los tipos de datos del lenguaje PASCAL tenemos el TIPO REAL. Su estudio es vital para entender como trabajar con lo números correctamente.

Seguimos este mes con el curso de Pascal. Por desgracia el mes pasado, un error hizo que nuestro estudio se viese truncado, ya que no apareció parte del artículo, pero este mes lo podéis encontrar en la página 66 de este número. Lo sentimos de corazón.

Comenzaremos viendo otro tipo de datos: El tipo REAL. Un número real es aquel que tiene una parte entera y otra fraccionaria. Por ejemplo, el numero PI (3.1416) tiene una parte entera (3) y una parte fraccionaria (.1416). Si le echas una ojeada al programa de cálculo de áreas del mes pasado, advertirás que la división la efectuábamos con DIV, era para números enteros.

Bien pero ¿y si hubiéramos puesto números reales en las variables? Pruébalo. por ejemplo sustituye las variables por las siguientes:

```
BASE := 10.3; ALTURA := 20.7;
```

Ahora intenta compilarlo (ALT+F9). Verás que TPascal nos avisa con un error. Esto es debido a que las variables BASE y ALTURA se declararon como enteras, es decir, que no contendrían números con parte fraccionaria. Cambia las líneas donde se declaran las variables y sustituye los INTEGER por la palabra REAL.

```
BASE : real; ALTURA : real
;RESULTADO : real;
```

Si ahora intentas compilarlo, verás que el compilador te vuelve a dar un error. ¿qué pasa ahora? Bien, si recuerdas el capítulo pasado, dimos los operadores para las operaciones básicas (+,-,*.DIV), pero dijimos que eran para números enteros. Los números reales, trabajan con los mismos operadores que los enteros, con la salvedad de la división, que en vez de DIV utiliza el símbolo '/'. Sustituye DIV por / y vuelve a compilar. Ahora si, todo va bien, pero... esto de usar DIV o

"/" según sea el tipo de la variable es muy pesado. No te preocupes, el operador "/" funciona tanto con enteros como con reales, así que podrás usar siempre el operador "/" sin importar el tipo de la variable. Eso sí, recuerda que si trabajas con números enteros, el operador DIV es más rápido que "/".

Echemos ahora un vistazo al resultado que nos da el programa. Si introdujiste los valores que indiqué antes en las variables BASE y ALTURA, el resultado de la operación habrá sido algo como:

```
1.0660500000E+02
```

¿qué es esto? Si no estas acostumbrado te sonará raro, esto es porque TPascal presenta los números reales en notación científica. El formato general de ésta notación es:

```
Número E+n
```

que se traduciría como Numero*10^n. Es decir el número que nos dió TPascal equivale a:

```
1.0660500000*10^2
```

Esta notación es muy útil, ya que nos permite usar números muy grandes sin que tengamos que llenar toda la pantalla con ellos. Sin embargo, no es una notación muy bonita para un programa de este tipo. Por suerte, el procedimiento WRITELN tiene opción para formatear la salida. Podemos decirle el número de decimales que queremos presentar y el número de espacios que ocupará el número en la pantalla. Para ello, sustituye la línea que tiene el procedimiento WRITELN por la siguiente:

```
WRITELN('El resultado es: ',resultado:1:4);
```

y vuelve a ejecutar el programa. Ahora verás como el número es presentado con cuatro decimales. Adivinarás que el número que señala el número de decimales es el segundo que hay tras la variable RESULTADO. El primer número señala el espacio mínimo que ocupara el número.

Para verlo mejor, sustituye el número 1 por el 15 (RESULTADO:15:4) y ejecútalo. Verás como aparece el número anterior, pero esta vez esta desplazado hacia la derecha. Esto es muy útil para justificar los número hacia la derecha, por ejemplo, si queremos presentar una ristra de números y queremos hacer que todos estén "pegados" a la derecha para, por ejemplo, mostrar una suma, sólo tenemos que poner como primer número el tamaño el mayor de ellos. El formato general para justificar un número, es:

```
variable:ESPACIOS:DECIMALES
```

MAS OPERACIONES CON ENTEROS

Hemos visto que TPascal puede hacer sumas, restas, multiplicaciones y divisiones. Pero para cualquier trabajo serio, estas operaciones no nos bastan. Veremos ahora algunas operaciones que podemos usar con números. Primero nos ocuparemos de los enteros (INTEGER). Además de las operaciones de suma, resta, multiplicación y división, TPascal posee otras. La primera de ellas es la operación MOD. Esta operación devuelve el resto (módulo) de una división. Su formato general es:

```
Resto := Num1 MOD Num2;
```

Por ejemplo, si queremos saber el resto de la división 13/4 y almacenarla en la variable RESTO, haríamos lo siguiente:

```
resto := 13 MOD 4;
```

Cuando se ejecute esta línea, la variable RESTO, valdrá 1, que es el resto de la división 13/4.

Entramos ahora en una parte algo espinosa de los números enteros, sin embargo, es muy necesaria, sobre todo en el campo de la programación de videojuegos. Nos referimos a operaciones a nivel de bits. Como sabes, el ordenador representa internamente los números como una ristra de ceros y unos. Pues bien, un número entero, en TPascal, tiene 16 bits o lo que es lo mismo 2 bytes.

Está fuera de los objetivos de éste curso la explicación a fondo de la aritmética binaria, así como la conversión de números de decimal a binario, Sin embargo esto ya se ha visto en el curso de ensamblador al hablar sobre el álgebra de Bool. La primera de las operaciones que vamos a ver es AND. Esta operación realiza un Y lógico entre dos números enteros. Si operamos 2 bits, el resultado es 1 si y sólo si los dos bits operados son 1 (figura 1). Veamos un ejemplo:

```
resultado := 44 AND 53;
```

En nuestro ejemplo RESULTADO valdría 36. Veamos porque.

```
44 = 00101100 53 = 00110101
AND 36 = 00100100
```

El siguiente es el operador OR. Este realiza un O lógico, es decir al operar dos bits, el resultado es 1 si cualquiera o ambos de los dos bits operados vale 1 (figura 2). Veamos un ejemplo con los mismos números anteriores:

```
resultado := 24 OR 53;
```

Ahora RESULTADO vale 61:

```
44 = 00101100 53 = 00110101
OR 61 = 00111101
```

Le toca el turno al OR eXclusivo. Al operar dos bits con XOR, el resultado es 1 si y sólo si uno de los operandos es 1, es decir, si ambos operandos son 1, el resultado es 0 (figura 3). He aquí el ejemplo:

```
resultado := 24 XOR 53;
```

RESULTADO vale ahora 25:

```
44 = 00101100 53 = 00110101
XOR 25 = 00011001
```

El próximo mes, más.

Tabla OR

```
1 or 1 = 1
1 or 0 = 1
0 or 0 = 0
0 or 1 = 1
```

Figura 1

Tabla AND

```
1 and 1 = 1
1 and 0 = 0
0 and 0 = 0
0 and 1 = 0
```

Figura 2

Tabla XOR

```
1 and 1 = 1
1 and 0 = 0
0 and 0 = 0
0 and 1 = 0
```

Figura 3

Una nueva sección para el ordenador AMIGA

Lenguaje Amos

¿QUE ES EL AMOS?

El AMOS es un LENGUAJE de PROGRAMACION de ALTO NIVEL con màs de 500 comandos, basado principalmente en el lenguaje BASIC pero que tiene unas caracterìsticas muy particulares que hacen de èl un lenguaje de programaciòn con el que, fàcilmente y sin meternos en las complicaciones que podrìa acarrear el tener que empezar desde cero a programar en C ò en ENSAMBLADOR, logramos unos resultados màs que sorprendentes en cuanto a velocidad y ejecuciòn.

Cualquiera que tuviera unos conocimientos mìnimos de BASIC y que dispusiera del tiempo suficiente, podría programar un VIDEOJUEGO. Este lenguaje està exclusivamente pensado para los poseedores del AMIGA de COMODORE. Lo ha escrito un señor llamado FRANÇOIS LIONET. Para mì, es uno de los mejores y màs versàtiles Lenguajes de alto nivel pensado para vosotros, a los que cierto gusanillo curioso alguna vez ha picado y os habéis preguntado, ¿por què yo no serìa capaz de hacer esto?. Con este lenguaje podréis hacer de principio a fin y paso a paso, eso si, como todo en este mundillo, con mucha paciencia, cualquier tipo de videojuego. (Ya sabéis todo en esto depende de vosotros mismos y de vuestra imaginaciòn). No me extiendo màs.

A lo largo de estos artìculos iréis comprobando por vosotros mismos todo lo que os he dicho.

ENTORNO DE TRABAJO

Para trabajar con el AMOS es necesario que tengais un entorno de trabajo, es decir: todos los elementos necesarios pensados ùnica y exclusivamente para conseguir las metas que os habéis propuesto. Para ello necesitaréis : un programa de dibujo (en este caso cualquiera de los DELUXE PAINT os serviria) pero si teneis el III, mucho mejor, pues podréis comprobar las animaciones y movimientos de los personajes antes de introducir los datos por el editor y asì os libraréis de tener que editar otra vez los gràficos para retocarlos.

Necesitaréis ademàs un programa de Mùsica. Servirà para adornar vuestras creaciones con la mejor mùsica que sepáis hacer. Os aconsejarìa dos (en Inglaterra son de dominio pùblico y al cambio no os costarìan demasiado) estos son: El G.M.C y el SOUNDTRAKER, pues los instrumentos de uno son compatibles con los del otro; ademàs el AMOS es totalmente compatible con los ficheros de mùsica creados con estos dos programas.

Eso sì, sin dejar aparte el AMOS.

MESA DE TRABAJO

Una vez que arranquéis el AMOS, veréis lo que es el Editor, desde el cual vosotros haréis todas las operaciones necesarias para ir llevando a buen fin vuestras ideas màs brillantes ; pero hablaremos de ello màs adelante. Por ahora no os dirè nada màs. En este artìculo, simplemente os he hecho una leve introducciòn de lo que es esta genial idea para nuestro COMODORE AMIGA.

AVANCE DE TEMAS.-

Durante este curso os irè enseñando, una serie de comandos, asì como a aplicarlos en una serie de ejemplos. Entre otras cosas, aprenderéis a:

- Diferenciar y usar los principales elementos gràficos de un videojuego para AMIGA: PANTALLA, ICONOS, BOBS, SPRITES.
- Usar los diferentes modos de pantalla y resoluciones.
- Manejar los diferentes bancos de memoria.
- Manejar el AMAL (sistema de animaciòn del AMOS). - Como editar sonidos. - Como crear mùsica y trasladarla al programa. - Como diseñar lo que serìan los mapas del juego. - Como usar las màscaras en los gràficos. - y un largo etc...

J.P.FREIRE

Los ojos del ordenador:

Aprende a manejar la tarjeta



El periférico más carismático de todo el hardware del PC es, sin lugar a dudas, la tarjeta de vídeo VGA. De su correcta programación va a depender que nuestro juego sea calificado como extraordinario o simplemente "pasable", incluso cuando la adicción del mismo sea grande.

CODIGO JUEGO

Continuamos el estudio de BIOS de vídeo o VBIOS.

Ya sabemos que éste se encuentra en la memoria ROM de la tarjeta de vídeo que insertamos en una de las ranuras de ampliación del PC. Gracias a esta movilidad, cada fabricante de tarjetas puede incluir sus propias rutinas de manejo de las mismas, de forma que incluso proviniendo de diferentes fabricantes, todas son compatibles a los ojos del programador.

También vimos el mes pasado como podemos hacer uso de dichas rutinas para nuestros propios propósitos, mediante el uso de la interrupción general de los servicios del VBIOS. Estamos hablando de 10h, o 16 decimal.

Por medio de ésta, se accede a los diferentes servicios estándar de vídeo. Algunos de ellos son: Establecer modo de vídeo, establecer forma del cursor, obtener forma del cursor, etc. Pero empecemos a verlas una a una.

INT 10h, AH=0, AL=modo
Establecer modo de vídeo

La primera de todas, y la más importante, es la que permite cambiar el modo de vídeo. Es decir, establecer los parámetros esenciales de la visualización en pantalla, los cuales son:

1) Tipo de información: Hay dos modalidades Gráfica y Alfanumérica. Indica la forma en que el hardware de la tarjeta mirará la información que está la VRAM en ese momento. Si el modo elegido es gráfico, los bytes de la información se tratarán como pixels o como parte de ellos. Sin embargo, si el modo escogido es alfanuméri-

co, los datos se tomarán como caracteres ASCII y sus consiguientes atributos de color. En los juegos, los modos utilizados por excelencia son el 13h y el 13X, el segundo de los cuales no es estándar y por lo tanto requiere una programación previa de los registros de la VGA, que estudiaremos más adelante.

2) Resolución. Este parámetro indica cual es el cantidad de información que podemos ver en la pantalla estática, es decir en un determinado momento. Siempre se mide en columnasxfilas. Si el modo gráfico es alfanumérico la unidad serán *caracteres*, si por el contrario es gráfico la unidad serán los pixels. De esta forma tenemos modos de 80 x 25 caracteres y de 640 x 480 pixels.

3) Número de colores. Es decir, cuántos colores podemos ver simultáneamente en pantalla. Estos van desde 4 en la tarjeta CGA, hasta los 16 millones de la SuperVGA.

Cuando llamamos a esta función deberemos cargar el registro AH con cero, lo cual indicará a qué función nos estamos refiriendo dentro del conjunto de la interrupción 10h. El registro AL, se carga con el modo de vídeo que queremos establecer y finalmente llamamos a INT 10h.

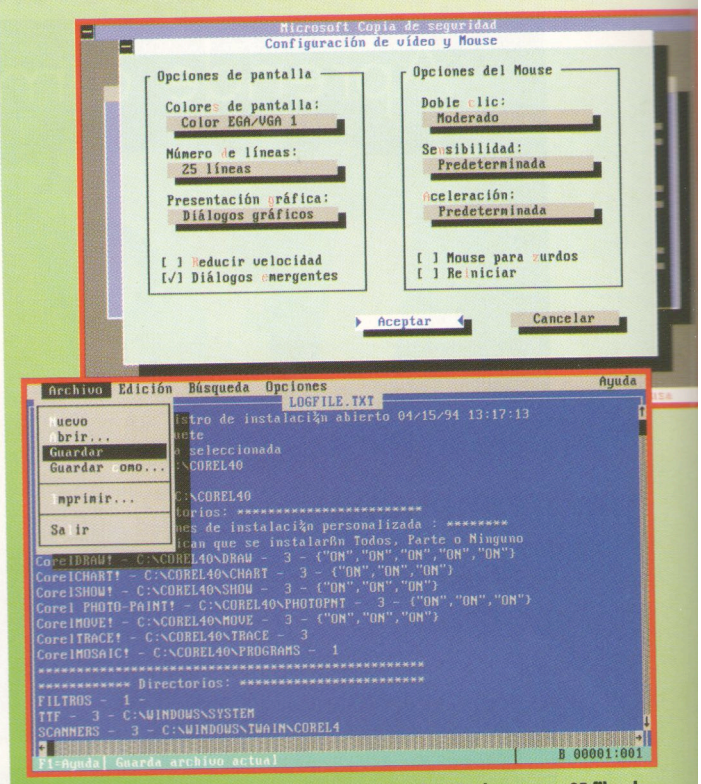
El sistema realiza una programación de los registros del hardware de la VGA de forma que se ajustan los parámetros para la nueva visualización. Gracias a ello, la VGA se dice que emula las tarjetas de vídeo anteriores a ella, como la MDA, CGA y EGA, ya que sitúa su hardware en las mismas condiciones que dichos adaptadores originales. Un truco muy útil, por cierto.

Grabados en ROM se encuentran los parámetros correspondientes a cada modo gráfico, por lo que la INT 10h tan sólo los tomará e introducirá en los puertos a los que están conectados los puertos del hardware de la VGA, mediante instrucciones OUT del lenguaje ensamblador.

Pero veamos los modos gráficos más usuales dentro de la VGA estándar.

El primero sería en el que entra el ordenador cuando lo arrancamos. El modo 03h (ver figura y pantallas adjuntas).

En él, no se logra una gran riqueza gráfica, ya que los únicos pseudográficos que podemos utilizar son los que se incluyen en el juego estándar del formato ASCII. Con estos, se pueden hacer gráficos por bloques, que aunque no nos sirvan de mucho para los juegos, si permiten hacer alguna que otra "virguería" como las que se ven en muchos otros programas de aplicación.



El modo 03h es en el que el ordenador arranca. Permite 80 columnas por 25 filas de caracteres, así como 16 colores simultáneos en pantalla.

Código	Modo
00h	Texto, 40x25, 16 colores
01h	Texto, 40x25, 16 colores
02h	Texto, 80x25, 16 colores
03h	Texto, 80x25, 16 colores
04h	Gráfico, 320x200, 4 colores
05h	Gráfico, 320x200, 4 colores
06h	Gráfico, 640x200, 2 colores
07h	Texto, 80x25, monocromo
0Dh	Gráfico, 320x200, 16 colores
0Eh	Gráfico, 640x200, 16 colores
0Fh	Gráfico, 640x350, monocromo
10h	Gráfico, 640x350, 16 colores
11h	Gráfico, 640x480, 2 colores
12h	Gráfico, 640x480, 16 colores
13h	Gráfico, 320x200, 256 colores

El uso de estos gráficos está muy extendido en la creación de cajas y ventanas, así como en menús desplegables como los que podemos ver en el EDIT que viene con nuestro MSDOS.

En este modo tenemos 80 columnas de caracteres y 25 filas. Para la mayoría de las aplicaciones esta cantidad de información simultanea en pantalla ha resultado suficiente. No así para los juegos, donde la vistosidad gráfica de los mismos ha impuesto la utilización de resoluciones más avanzadas.***

Programación del Hardware del AMIGA

Por:

Alvaro Mateos

El Amiga de Commodore es un ordenador, que como plataforma para los juegos, resulta una máquina maravillosa.
De todos los aspectos que engloba la programación en este sistema, la correcta manipulación de hardware es muy importante.

El AMIGA es un ordenador que, aunque en su configuración básica (Amiga 500) posea un microprocesador M68000 a tan sólo 7 MHz sorprende por su capacidad en el tratamiento de gráficos que lo convierte en una máquina perfecta para el desarrollo de videojuegos.
Su secreto reside en sus coprocesadores de apoyoque liberan a la CPU de ciertas tareas específicas que el microprocesador tardaría mucho más tiempo en ejecutar.
Un microprocesador es un chip de propósito general. Esto significaque está diseñado para realizar cualquier tipo de tarea; así, tenemos instrucciones lógicas, aritméticas, de transferencia de datos, etc. Pero es difícil pensar que todas estas instrucciones las realizaría más lentamente que un chip que estuviese especializado en ejecutar un tipo concreto de instrucciones. Estos son los chip de propósito específico, y como ejemplo tenemos un coprocesador matemático que realiza operaciones matemáticas que el microprocesador por sí solo tardaría variasveces más en ejecutar. En el AMIGA tenemos lo que podríamos llamar coprocesadores gráficos, y la correcta programación de éstos es lo que hace que nos sorprendamos cuando nos sentamos ante un juego con scroll multidireccional y múltiples gráficos moviéndose a la vez. Estos coprocesadores se conocen como los "custom chips"que fueron diseñados a medida para el AMIGA. Lo componen tres chips, llamados AGNUS, DENISE Y PAULA y veremos a continuaciónuna breve descripción de ellos.

AGNUS

Este chip es el más importante, ya que los otros dos dependende él. Por una parte controla el DMA. El DMA (Direct Memory Access,Acceso Directo a Memoria) es un sistema por el cual los dispositivos se conectan directamente a la memoria, pudiendo escribir/leer datos a/de la memoria sin necesidad de molestar a la CPU. Esto es bastante útil si tenemos en cuenta que los dispositivos necesitan acceder continuamente a la memoria, y sin la ayuda del DMA estarán constantemente interrumpiendo a laCPU, sin dejarle tiempo para su principal tarea: ejecutar el proceso en curso. Por otra parte, el AGNUS tiene una circuitería que permite la transferencia de memoria a memoria de una forma muy especial, mediante el llamado BLITTER. El BLITTER es el responsable de imprimir los móviles en pantalla a gran velocidad,

pudiendo enmascarar gráficos y mezclarlos con el fondo, consiguiendo que se muevan por el escenario sin corromperlo. Otra característica del AGNUS es su capacidad de sincronizar ciertas operaciones con el barrido de pantalla. Para entender ésto, imaginemos que podemos ver el funcionamiento de nuestro monitor "a cámara lenta"; veríamos así como se van coloreando los diferentes puntos de la pantalla de izquierda a derecha(hasta completar una línea) y de arriba a abajo (hasta completar un "frame" o cuadro). Esto ocurre 50 veces por segundo y se denomina refresco o barrido de pantalla. Pues bien, el AGNUS sabe en cada momento donde se encuentra el puntero del "raster" (haz de electrones) y podremos programarlo para que haga lo que queramos cuando el raster se encuentre en un punto determinado. Por ejemplo, podemos cambiar el modo de vídeo cuando se alcancen las 20 últimas líneas de pantalla, creando simultaneamente dos modos de vídeo diferentes, uno para la pantalla de juegos y otro para el marcador. Esto podemos hacerlo programando el COPPER, que trataremos más adelante.

DENISE

Este chip es el encargado de las señales de vídeo del AMIGA. El AGNUS, mediante DMA, va cogiendo los datos de la memoria y enviándolos a este chip que, dependiendo del modo en que esté programado, procesará estos datos para dar lugar a la imágen de vídeo que veremos en nuestro monitor. Posée varios modos de pantalla, dos tipos de resolución(alta y baja), una paleta de 4096 colores y 32 registros de color. Permite el uso de sprites, scroll por hardware y varias características más, que ya comentaremos.

PAULA

Este no es un chip gráfico, sino más bien un chip diseñado para la gestión de entrada/salida de los periféricos, pero posée también un convertidor digital/analógico encargado de los cuatro canales de sonido del AMIGA. Además es el encargado de gestionar las interrupciones del sistema, incluídas las generadas por los dos chips anteriores.

PROGRAMACION DE LOS CUSTOM CHIPS

Tras describir brevemente los chips AGNUS, DENISE Y PAULA,veamos cómo podemos programarlos según nuestras necesidades.

Cada uno de los chips tiene asociado una serie de registros que estarán localizados en distintas posiciones de memoria. A la hora de programar los chips no nos hará falta saber de cualde ellos se trata porque eso será transparente para nosotros. Unicamente nos interesa saber qué hace cada registro.

Como hemos dicho, un registro será para nosotros una posición de memoria, en la que según el valor que escribamos estaremos programando el chip de una forma u otra. Hay también registros de lectura que nos servirán para conocer el estado de los chips. Todos estos registros están localizados a partir de la dirección $dff000 (en hexadecimal) y nos referiremos a cada uno de ellos con el offset correspondiente a partir de esa posición.

Vamos a poner un ejemplo sencillo de programación de estos registros. Existen 32 registros de color nombrados como COLOR00 ... COLOR31, que están localizados a partir de ladirección $dff180 (offset $180). Los registros de color responden al sistema RGB, pudiendo elegir 16 tonalidades diferentes de rojo, verde y azul para crear el color deseado.Tendremos por tanto 16 elevado a 3 = 4096 colores diferentes. En el siguiente programa (Listado 1) iremos cambiando los dos primeros registros de color (COLOR00 y COLOR01) en un bucle cerrado. En la pantalla veremos, como podeis imaginar, un arcoiris de colores.

EL COPPER

Como explicamos antes, el COPPER es un pequeño microprocesador, dentro de AGNUS, que está sincronizado con el barrido de pantalla. Lo que este

```
;       PROGRAMACION DEL HARDWARE DEL AMIGA.;
;       Listado 1
CUSTOMS     equ $dff000
COLOR00     equ $180
COLOR01     equ $182
Inicio: lea     CUSTOMS,a0      ; apuntamos con a0 la base de los registros
        move    #$0000,d0               ; cargamos d0 con el color negro
        move    #$0fff,d1       ; cargamos d1 con el color blanco
        loop
        move    d0,COLOR00(a0) ; escribimos en el registro del color 0
        move    d1,COLOR01(a0) ; escribimos en el registro del color 1
        addq    #1,d0           ; cambiamos el color simplemente
        and     #$0fff,d0       ; incrementandolo, ajustandolo a los
        subq    #1,d1           ; 4096 colores posibles
        and     #$0fff,d1
        btst.b  #6,$bfe001      ; mientras no pulsemos el boton del raton
        bne     loop            ; seguimos en el bucle
```

Listado 1

El amiga viene equipado con una serie de coprocesadores que ayudan en gran medida a la liberación del microprocesador de ciertas tareas, con lo que, y a pesar de ser lento (7mhz) permite manejar sonido y gráficos con gran fluidez

microprocesador nos permite hacer es, básicamente, esperar a que el raster alcance una posición determinada y escribir en cualquier registro de los custom chips. Decimos que es como un pequeño microprocesador porque, como tal, coge instrucciones de la memoria y las ejecuta. Las instrucciones que puede realizar son tan sólo tres: MOVE, WAITy SKIP.

MOVE registro,valor:

Esta instrucción es como la del 68000 donde 'valor' es un dato de 16 bits y registro es el offset de16 bits que indica en cuál de los registros de los custom chips se va a escribir. Esto nos permitirá cambiar el modo de pantalla, cambiar la paleta de colores o manejar comodamente los sprites, entre otras cosas. Podemos también generar la interrupción que deseemos escribiendo en el registro INTREQ. Esta posibilidad es bastante útil si queremos evitar parpadeos a la hora de imprimir gráficos. Así pués, podemos generar una interrupción cuando el raster esté al final de nuestra pantalla de juego para empezar la impresión de gráficos sin que el raster nos atrape a mitad del proceso, lo que produciría distorsión enla imagen, e incluso un parpadeo.

WAIT vp,hp:

Con esta instrucción le decimos al COPPER que espere hasta que el raster llegue a la posición vp,hp de lapantalla. Cuando el COPPER encuentra esta instrucción, pueden ocurrir dos cosas:

- Si el raster se encuentra en una posición anterior a ésta, espera a que llegue, y cuando esto ocurre, coge la siguiente instrucción y la ejecuta.

- Si el raster se encuentra en una posición posterior, entenderá que no debe esperar algo que ya ocurrido y cogerá la siguiente instrucción.

SKIP vp,hp:

Actúa de forma parecida a WAIT, sólo que cuando se alcanza una posición dada, se salta la instrucción que viene a continuación y ejecuta la siguiente. Esta instrucción se usa normalmente para realizar bucles.

FORMATO DE LAS INSTRUCCIONES

Las tres instrucciones anteriores están formadas por dos palabras de 16 bits y su formato es el siguiente:
MOVE : La primera palabra contiene el offset del registro a escribir, y la segunda el dato de 16 bits que será transferido al registro.
Primera palabra: - bit 0 siempre a 0 - bits 1-8 offset indicando el registro - bits 1-15 no se usan, pero deben ponerse a 0 Segunda palabra: - bits 0-15 dato a escribir Ejemplo: dc.w $0180,$0FFF ; pone en el registro COLOR00 (offset $180); el valor $0FFF (color blanco)
WAIT : La primera palabra contiene la posición horizontal y vertical del raster. La segunda actúa como una máscara, que indica qué bits de la posición horizontal y vertical deben tomarse en cuenta.
Primera palabra: - bit 0 siempre a 1 - bits 1-7 posición horizontal - bits 8-15 posición vertical. Segunda palabra: - bit 0 siempre a 0 - bits 1-7 máscara horizontal - bits 8-14 máscara vertical - bit 15 poner a 1 Por ejemplo, para esperar a la posicion 100 ($64 en hex.) vertical:dc.w $6401,$FF00 ; espera a la linea 100 sin importar la coordenada horizontal

SKIP. Igual que WAIT, sólo que el bit de la segunda palabra debe valer 1.

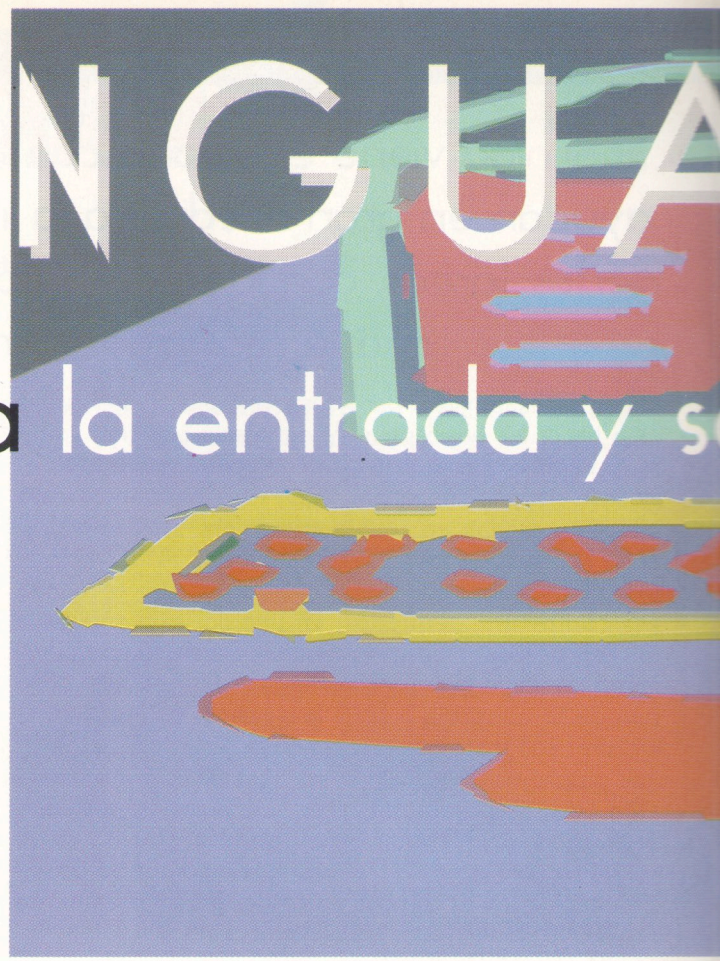
PROGRAMACION DEL COPPER

Para programar el COPPER tenemos que escribir la "copper list", esto es, la serie de instrucciones que queremos que ejecute. Como no tenemos ensamblador para el COPPER, tendremos que codificar las instrucciones manualmente y escribirlas en el ensamblador del 68000, como si fueran datos (con dc.w). Al final escribiremos la instrucción WAIT 255, 127 (dc.w $FFFF, $FFFE), que indicará el final de la "copper list". Una vez escrita y ubicada en la memoria, debemos indicar al COPPER dónde debe empezar a ejecutar las intrucciones,escribiendo en el registro COP1LC la dirección de nuestra "copper list". Cada vez que empiece el barrido de pantalla, el COPPER buscará en este registro la dirección donde comienzan las instrucciones que debe ejecutar. La "copper list" se ejecutará desde el principio una vez cada 1/50 de segundo. De todas formas existe otro registro para forzar al COPPER a saltar a la primera instrucción. Este registro es el COPJMP1, y basta con escribir en ól cualquier valor para que el COPPER salte a la primera instrucción. Existen además otro par de registros, COP2LC y COPJMP2, que actúan de igual forma y que se usan en combinación con la instrucción SKIP para realizar bucles. Aunque todo lo dicho anteriormente puede resultar bastante complicado, vamos a ver un ejemplo que demostrará lo fácil que es escribir una lista de instrucciones para el COPPER. En el siguiente ejemplo (Listado 2) creamos un modo de pantalla con un sólo color, pero con la ayuda del COPPER realizaremos un truco que permitirá que se vean en pantalla cinco colores diferentes. Tema éste último, motivo de nuestro próximo artículo.

CURSO

LENGUA

Por:
José Antonio Brihuega



Introducción a la entrada y s

En el número anterior pre sentábamos un pequeño programa, llamado lista.c, el cual reproducimos en la figura número 1.

Si el lector intentó ejecutarlo, se encontraría que lo único que hace es sacar por pantalla la serie de caracteres ASCII, del 33 hasta el 254.

La variable 'carácter' toma el valor inicial 33, que va incrementándose mientras sea menor que 255. A cada vuelta, se imprime el carácter correspondiente (función putchar).

La función putchar() no hace salto de línea, por lo que los caracteres van saliendo juntos. Al terminar el bucle for, el cursor se encuentra justo tras el último carácter. La instrucción putchar('\n') da un salto de línea para que el cursor pase al principio de la siguiente línea. Así logramos que el "prompt" del sistema salga un poco separado al terminar el programa.

El '\n' es la forma que C tiene para representar el salto de línea. Los caracteres especiales se escriben siempre con una barra invertida y una letra. '\r' es el retorno de carro (el cursor vuelve al extremo izquierdo sin saltar de línea); '\t' es el tabulador. Cuando queramos representar una barra inver tida propiamente dicha, la pondremos doble ('\\').

La barra invertida seguida por un número de hasta tres cifras se interpreta como el código ASCII de un carácter. El número debe ser octal (base 8). El '\0' (cará cter nulo) se interpreta como fin de cadena.

ENTEROS COMO CARACTERES

Hay una serie de características del programa que lo hacen interesante.

En primer lugar, no tiene mucha lógica usar una variable int para representar caracteres. Esto deja de extrañar al saber que la función putchar() requiere un int como argumento. En general, todas las funciones de entrada/salida estándar representan los caracteres como int en vez de char.

De todas formas, teniendo en cuenta que, internamente, los char se tratan como enteros, no suele haber problemas pasándole un char a una de estas funciones. La verdadera razón por la que en este programa se declaró 'carácter' como int es un poco menos "obvia".

Pruébese a cambiar int por char en la declaración de 'carácter':

```
char caracter;
```

¿Qué ocurre al ejecutar el programa con este cambio? Nuestro programa no termi na nunca, repitiendo infinitamente la lista ASCII. ¿Qué ha pasado?

Al comienzo de este curso dijimos que el rango de valores de cada tipo depende del compilador que usemos. Sin embargo, la mayoría suelen coincidir en muchos casos: El tipo char toma valores de menos 128 a 127, mientras que el int lo hace de -32768 a 32767.

Los tipos funcionan cíclicamente, es decir, al sobrepasar el 127 en tipo char, se vuelve a empezar por el - 128.

Sabiendo esto, resulta obvio que la condición

```
caracter < 255
```

siempre será verdadera, y el bucle for se hace infinito. Si definimos 'carácter' como int, se evita este problema.

Por cierto, puede resultar extraño que el tipo carácter tome valores negativos, pero no lo es tanto sabiendo que C trata los caracteres como números enteros. Siendo así, es lógico también que exista el tipo "unsigned char", que toma valores de 0 a 255. ¿Serviría este tipo para usarlo en nuestro programa lista.c?

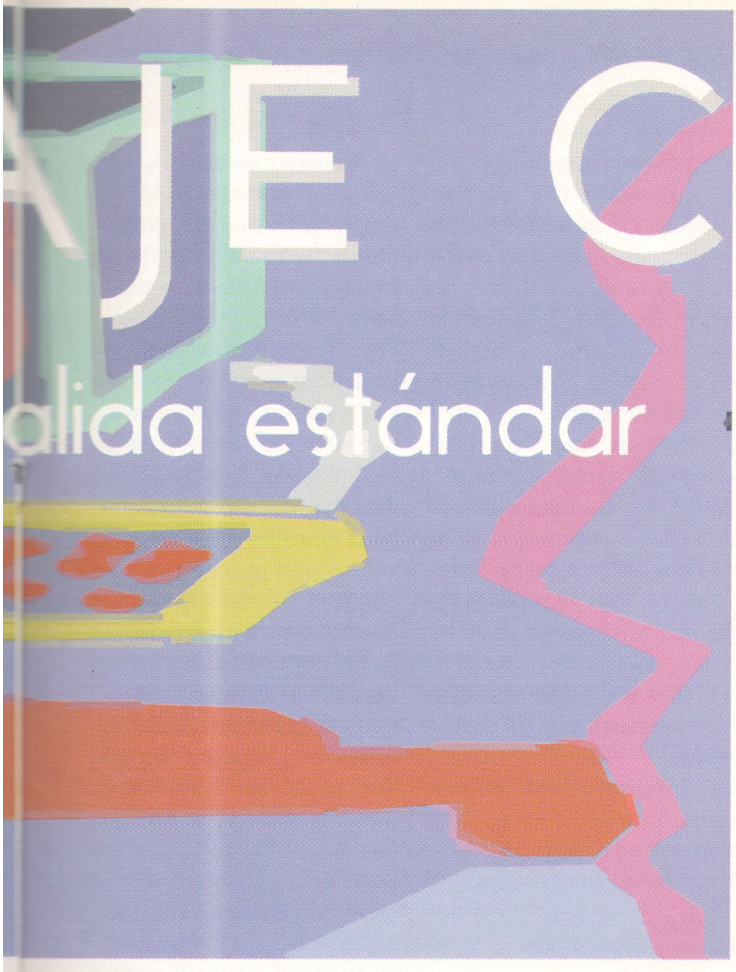
LA ORDEN #INCLUDE

Hasta ahora habíamos su puesto que las funciones usadas por un programa tenían que definirse dentro del mismo; es decir, que cuando invoquemos la función "fun()", en algún otro lugar del programa tendría mos que decirle al compila dor qué hace exactamente fun().

Sin embargo, esto no es siempre así, pues el propio compilador posee una serie de bibliotecas (o "librerías") de funciones predefinidas.

Además, nosotros mismos podemos definir las nuestras, o incluso comprar algunas de las que se comercializan.

Cada compilador puede traer cientos de funciones predefinidas que nos pueden ahorrar




```
/*LISTA.C - Lista de caracteres ASCII*/

#include <stdio.h>

int main(void) /*Cabecera de funcion*/
{
  int caracter;  /*Declaracion de variable
local*/

  for (caracter = 33 ; caracter < 255 ;
caracter++)
    putchar(caracter);

  putchar('\n');

  return 0;
} /*fin de main()*/
```

Figura 1, LISTA.C

```
/*MAYUS.C – Programa que convierte en pun-
tos las letras mayúsculas*/

#include <stdio.h>    /*Biblioteca estándar
de E/S*/

int main(void)        /*Cabecera de main*/
{
  int letra;          /*Variable local*/

  while ((letra=getchar()) != EOF)
    if (letra>='A' && letra<='Z')
      putchar('.');
    else
      putchar(letra);

  return 0;
} /*fin de main*/
```

Figura 2, MAYUS.C

mucho trabajo. Lo malo es que pueden ser diferentes de uno a otro compilador. Las únicas que seguro están en todos son las que se llaman "bibliotecas estándar".

Si usamos estas bibliotecas, tendremos la ventaja de que nuestros programas serán compatibles con todos los compiladores, incluso los que corren en diferentes sistemas operativos. A esta característica se le llama "Portabilidad".

Para usar funciones de una biblioteca, ya sea estándar o no, tenemos que incluir la declaración de sus funciones, constantes, estructuras, etc. en nuestro código, de manera que el compilador las reconozca cuando las usemos.

Todas estas declaraciones de encuentran en unos archivos especiales, llamados "de cabecera" (header files), que tienen extensión ".h". Cada biblioteca tiene uno o varios.

En diferentes manuales encontraremos referencias de las funciones de biblioteca, y los diferentes archivos de cabecera donde vienen declaradas. Las cabeceras estándar más comunes son las que vienen en la tabla adjunta.

Para incluir una cabecera (por ejemplo, "stdio.h") en uno de

CABECERAS DE BIBLIOTECA ESTANDAR MAS COMUNES

stdio.h	Funciones de entrada/salida
stdlib.h	Conversión de tipos y asignación de memoria
string.h	Manejo de cadenas de caracteres
ctype.h	Comprobaciones y conversiones de tipos
io.h	Control de disco a bajo nivel
math.h	Funciones matemáticas en punto flotante
dos.h	Interacción con el MS-DOS
dir.h	Búsqueda de ficheros y manejo de directorios

nuestros programas pondremos al principio:

#include <stdio.h>

Tanto el (#) como los ángulos deben ponerse tal y como aparecen. Entre el (#) y la palabra include no debe haber ningún espacio.

Los ángulos dicen que el archivo stdio.h debe buscar se en el directorio especialmente dedicado a cabeceras, y no en el actual. Para buscarlo en el directorio actual, lo pondremos entre comillas dobles, y no entre ángulos.

En nuestro ejemplo lista.c, usamos la función de librería estándar "putchar()", cuya declaración está en el ya nombrado stdio.h. Por ello, lo incluimos al princi pio.

LA FUNCIÓN MAIN

En la cabecera de la función main, el int dice que main devolverá un entero. Este entero lo usará el sistema operativo para saber cómo terminó el programa (ERRORLEVEL). Normalmente, devolverá cero cuan do termine normalmente, y un valor distinto de cero al ocurrir algún error.

El valor devuelto por cual quier función (incluida main) será el especificado en la sentencia return. Esta instrucción termina la ejecución, retornando el valor especificado. El valor se puede omitir, devolviendo en ese caso un valor indefinido. También se puede omitir toda la instrucción return, terminando entonces al llegar al final de la fun ción.

El void dentro de los parén tesis, indica que main no tomará ningún parámetro (tipo vacío).

Se pueden eliminar el int y el void, quedando:

main()

La forma larga tiene la ventaja de que es más explicativa, y unifica el formato de main con el del resto de las funciones. A efectos prácticos, el funcionamiento es idéntico.

Así pues, vemos que en nuestro programa lista.c se han incluido varias cosas que, aunque no sea obligatorio ponerlas, facilitan la lectura del código. Esta norma nos permitirá hacer programas más legibles y coherentes.

UN PROGRAMA FILTRO

El siguiente programa "mayus.c" es ejemplo de una clase de programas muy útil: los "filtros". Un programa filtro toma una secuencia (flujo o corriente) de caracteres de entrada, opera algún cambio sobre ellos, y los pone en la salida, como se observa en el esquema.

La "entrada" y la "salida" pueden ser ficheros o dispositivos. En este ejemplo, para simplificar, tomaremos la entrada del teclado (función getchar) y pondremos la salida en la pantalla (putchar). Ambas funciones están recogidas en la cabecera stdio.h, así que usaremos el correspondiente #include. Ver Figura 2.

La transformación que realiza mayus.c en la corriente de caracteres es la supresión de todas las letras mayúsculas, las cuales serán sustituidas por puntos. Las minúsculas y otros símbolos pasan el filtro sin modificación.

Tras el trámite de las cabeceras y declaraciones, el núcleo del programa es un bucle while, que termina al leer de la entrada un carácter de "final de fichero" (EOF: end of file).

EOF debe escribirse en mayúsculas, y es una constante numérica definida en stdio.h, la cual representa el final de la "corriente" de caracteres, ya sea proveniente de un fichero, o de un dispositivo, como en este caso.

Cada dispositivo tiene una forma de indicar el final, pero, a la hora de programar, el uso de la constante EOF nos vale en todos los casos. En el caso del teclado, el final se indica pulsando simultáneamente las teclas Control-Z. Entonces, la función getchar reporta el citado EOF, concluyendo nuestro bucle while.

Se ha mencionado que EOF es una constante "numéri ca". Está fuera del rango para el tipo char o unsigned char, lo que nos obliga a definir "letra" como int. Esto no supone problemas, pues, como ya se ha dicho, las funciones putchar, get char y similares manejan los caracteres por su código ASCII.

La condición que hemos puesto en el while es más complicada que las que hemos usado hasta ahora:

```
while ((letra=getchar()) != EOF)
```

Es un ejemplo de la potencia y flexibilidad de C, frente a otros lenguajes. La parte derecha de la expresión:

... != EOF

indica que se repita el bucle mientras que "algo" no sea el fin de la entrada. Pero, ¿qué es ese "algo"?

(letra=getchar())

Nos encontramos ante una expresión de asignación (no confundir con la compara ción ==). A la variable "letra" le asignamos el valor que nos proporcione la función getchar, que no es otro que el siguiente carácter pulsado.

Así pues, estamos leyendo del teclado, asignando la lectura a una variable e, inmediatamente, comparándola con la constante EOF. Todo ello en la misma línea de programa.

Ello es posible porque en C, al contrario de la mayoría de lenguajes, la asignación (=) no es una instrucción propiamente dicha, sino un operador, que puede usarse dentro de expresiones más complejas. El operador, no sólo da valor a una variable, sino que proporciona dicho valor para ser usado por el resto de la expresión.

La asignación tiene menor prioridad que la compara ción; por ello, ponemos los paréntesis forzando que se realice antes. Los paréntesis más externos son obligatorios para el while; los que están vacíos tras getchar indican que ésta es una función que no recibe parámetros. Tantos paréntesis anidados dificultan un poco la comprensión del código, pero este inconveniente se compensa con creces, pues esta expresión es más compacta y eficiente que cualquier otra que podamos usar. En C, una vez más, se prefiere la eficiencia a los formalismos.

Dentro del while, encontra mos una construcción "if...else", que no es necesario encerrar entre llaves, pues se trata como una unidad.

En la condición del if, volvemos a encontrar valores numéricos mezclados con caracteres. Las comillas simples identifican los litera les.

La expresión:

letra>='A'

comprueba si el valor ASCII contenido en "letra" (que es el que se leyó con getchar) es mayor (el código) o igual al de la letra A mayúscula. Esto es, si el carácter que se leyó está después de la A en la lista ASCII (o es la pro pia A).

Igualmente, la expresión:

letra<='Z'

comprueba si dicha letra es la Z mayúscula, o está antes de la Z en la lista ASCII. El operador && obliga a que ambas expresiones sean ciertas.

Resumiendo, si el carácter leído está entre la A y la Z mayúsculas, ambas inclusi ve, se escribe un punto en la pan-

talla, en otro caso (else) se escribe la propia letra.

La comprobación se repite indefinidamente, hasta que se pulse Control-Z, lo que hará que la condición del while sea falsa, terminando el programa (return). Resumimos todo ello en un diagrama de flujo, como el de la figura.

Si ponemos a funcionar el programa, veremos que getchar no funciona exactamente como pensábamos que lo haría.

En efecto, getchar no lee carácter a carácter a medida que tecleamos, sino que retiene una línea completa. Cuando pulsamos Intro, los caracteres de la línea se liberan uno a uno en cada ejecución de getchar. Una vez agotados los caracteres, el siguiente getchar volverá a retener una línea completa. Esto es así porque el teclado usa un "buffer" (memoria intermedia) donde se lee toda una línea. La función getchar toma los caracteres del buffer, no directamente del teclado.

Así pues, ponemos a funcionar el programa y tecleamos las siguientes líneas, pulsan do Intro tras cada una de ellas:

Dennis Ritchie creó el lenguaje C trabajando para la AT&T.

Lo que veremos en pantalla será una mezcla de lo que hemos tecleado y lo que el programa produce como salida:

Dennis Ritchie .ennis .itchie creó el lenguaje C creó el lenguaje . trabajando para la AT&T trabajando para la ..&.

Como esperábamos, el programa convierte cada mayúscula en un punto. Cuando queramos terminar, pulsaremos Control-Z (apa recerá en pantalla como ^Z), y aún tendremos que pulsar Intro una vez más para liberar el buffer.

CANALES ESTANDAR

Aunque los caracteres aparezcan en pantalla mezclados, en realidad llegan hasta ella por tres "canales" (en inglés "streams") diferentes:

a) La "entrada estándar" (que en C se denomina "stdin") es el canal del que toma los datos la función getchar y sus equivalentes. El sistema operativo, si no se le requiere otra cosa, conecta este canal con el teclado y, curiosamente, también con la pantalla. Se dice que los caracteres tecleados tienen "eco" a pantalla (por eso en el ejemplo anterior la entrada se mezclaba con la salida).

b) La "salida estándar" (stdout) es el canal en el que ponen sus datos las funciones putchar y similares. Por defecto, está conectado a la pantalla.

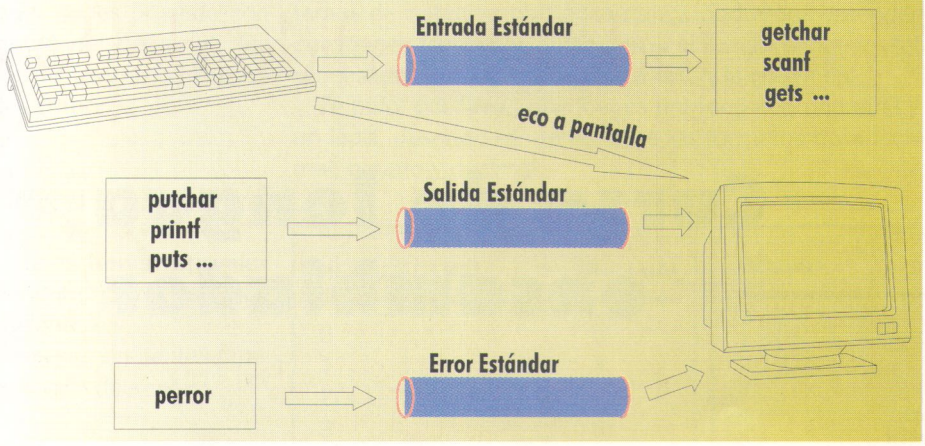
c) El canal de "error están dar" (stderr) está pensado para que los programas pongan en él sus mensajes de error, así como otras indicaciones que se quiera hacer al usuario. La diferencia con la salida estándar está en que stderr siempre está conectado a la pantalla. Es imposible, al menos en MS-DOS, reconducirlo a un fichero, a otro programa, etc., tal como haríamos con stdout. Usando el error estándar, nos aseguramos que nuestros mensajes de error nunca se van a "escapar" a otro sitio que no sea la pantalla.

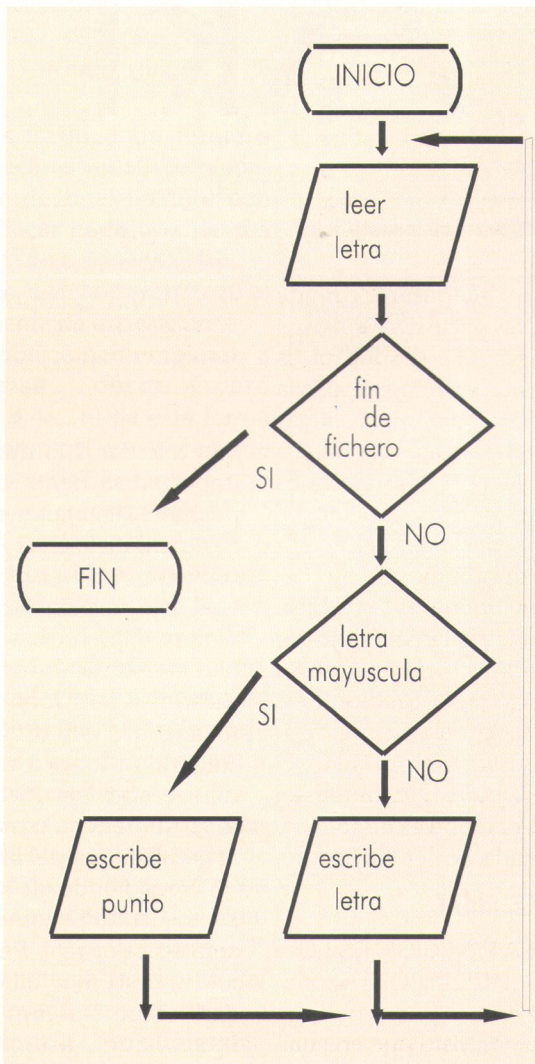
En el gráfico adjunto se representan estos canales como si se tratasen de tube rías. Aparecen con sus asignaciones por defecto, y ejemplos de funciones de stdio.h que hacen uso de ellos.

En un próximo número trataremos en profundidad la programación de estos canales estándar, y cómo aprovechar su capacidad de reconducirse a cualquier fichero, dispositivo e, inclu so, a otros programas.

Esta última cualidad nos permitirá crear fácilmente programas que cooperen entre sí.

ASIGNACION DE LOS CANALES DE ENTRADA/SALIDA ESTANDAR





4 parte de nuestro curso

Saca todo el partido de tu ordenador

Curso de lenguaje ensamblador Motorola 68000 para ordenadores ATARI



Por:

Andrés Lopetegui

Tenemos ya una leve idea de como el microprocesador es capaz de entender nuestras órdenes gracias al sistema binario. Los unos y los ceros constituyen el universo numérico del ordenador ir más allá sería entrar en los márgenes de lo humano.

BINARIO, CONTINUACIÓN

Como vimos, representar un número binario es útil para entender como el microprocesador "mira" los numeros. Resulta un sistema rústico y lioso, eso sí, pero nuestro ordenador es capaza de trabajar con el a muy alta velocidad, cosa que nos interesa. ¿Qué hacemos entonces si queremos trabajar con los números de forma más cómoda? La respuesta es sencilla pero complicada a la vez. Nos vemos obligados a traducirlos a otro sistema o notación donde su representación nos resulte más sencilla para el entendimiento humano. El sistema utilizado por excelencia es el Hexadecimal, por su correspondencia con el binario y su facilidad de traducción al sistema decimal. Este último sería el sistema ideal, ya que es el más cercano al hombre y su modo de ver los "numeros".

LA POSICION ES LO QUE IMPORTA

Hemos visto que en cualquier sistema numerico que utilizemos lo que más importancia tiene es el la posición que ocupe el dígito dentro de la cifra. Así, en el sistema decimal en el cual sólo contamos con 10 dígitos diferentes, se pueden construir números que pueden rondar entre menos infinito y mas infinito. Pues bien, lo mismo ocurre con los dos sistemas númericos restantes, el hexadecimal y el binario.

Dependiendo de donde se encuentre el número así será su valor real. Pero, en cada sistema, el valor variará en función de la base que utilicemos. Binario base 2, decimal base 10 y hexadecimal base 16.

En el decimal hallamos el exponente de diez. En el binario hallamos el exponente de dos y en hexadecimal el exponente de 16.

Por ejemplo:

Sistema decimal
Dígitos del 0 al 9

Posición 0....$10^0 = 1$
Posición 1....$10^1 = 10$
Posición 2....$10^2 = 100$
etc

Sistema binario
Dígitos 0 y 1

Posición 0....$2^0 = 1$



Posición 1....2^1=2
Posición 2....2^2=4
etc

Sistema Hexadecimal
Dígitos del 0 a la F

Posición 0....16^0=0
Posición 1....16^1=16
Posición 2....16^2=256
etc

Por lo tanto, según la posición que ocupen y el sistema de notación numérica así será el valor real de la cifra.

Por convención, a los números binarios se les añade una "b" al final, a los decimales "d" y a los hexadecimales se les precede con el símbolo del dólar "$". De esta forma usaremos números tales como:

12345d
$12345

los cuales tienen un valor real diferente, como veremos más adelante.

Lo números siguientes también son todo diferentes:

10011d
$10011
10011b

¿Os dáis cuenta de por qué?

SISTEMA HEXADECIMAL

Para traducir un número hexadecimal en su correpondiente decimal bastará con multiplicar cada una de sus posiciones por su valor, el cual se calcúla elevando 16 al exponente de la posición.

La posiciones empiezan a contarse desde cero (0,1,2,3,4,...,n) de esta forma obtendremos los valores que podéis ver el la tabla 1 que podéis ver en esta página.

Para averiguar el valor decimal, sumaremos el total de todas. Tabla 2.

La traducción de un número hexadecimal a binario es más sencilla, ya que cada dígito se puede representar por 4 bits. Tabla 3.

En el próximo número veremos las otras posibilidades de traducción, es decir, de decimal a hexadecimal y a binario.

Esta materia puede resultar algo pesada pero es de gran importancia. Sin un conocimiento exahustivo de los sistemas de notación numérica nunca podremos entender cuestiones tan importantes como el funcionamiento del procesador, lo chips de apoyo a éste y la memoria.

Te pido, por favor, que jueges un poco con un papel y un lápiz, pasando números entre los formatos hexadecimal, binario y decimal.

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

Valor decimal de los dígitos hexadecimales

Posición	valor
0	1
1	16
2	256
3	4096
4	65535
5	1048576
6	16777216
7	268435456

```
Ejemplo: $45F8
-------------

4096 x  4 = 16384
 256 x  5 =  1280
  16 x  F =   240
   1 x  8 =     8
          ---------

suma _____ 17912
```

Tablas 1 y 2

HEXADECIMAL	BINARIO
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Tabla 3

El microprocesador sólo es capaz de entender los ceros y los unos, luego tendremos que adaptarnos a su sistema de numeración.

4 parte de nuestro curso

Saca todo el partido de tu ordenador

Curso de lenguaje ensamblador Motorola 68000 para ordenadores AMIGA



Por:

Carlos Garrido Novoa

En el número anterior comenzamos a explicar el sistema de numeración decimal y lo retomamos ahora desde el punto donde lo dejamos.
Como ya hemos estudiado, en decimal el 10 es el menor número que no es representable en una sola casilla, lo que quiere decir que es el primer número que abarca la casilla inmediata a la izquierda y está compuesto por un cero y el menor número (un uno) en la casilla de la izquierda, la cual tiene un valor diez veces mayor que la anterior, y la siguiente diez veces más que esta, o sea 100, y así ...
Por lo tanto el número:

```
6 2 0 5 1 0
| | | | | \
| | | | |  - Casilla de valor 1
| | | |  \- Casilla de valor 10
| | | \- Casilla de valor 100
| | \- Casilla de valor 1.000
| \- Casilla de valor 10.000
\- Casilla de valor 100.000
```

Este número 620510 también se puede expresar de la siguiente forma:

```
     0 por 1 =      0
    1 por 10 =     10
   5 por 100 =    500
 0 por 1.000 =      0
2 por 10.000 =  20000
6 por 100.000 = 600000
-----------------------
SUMADO        = 620510
```

Todo esto te sonará mucho e incluso puede paracerte trivial, pero su importancia se pone de manifiesto a la hora de comprender el resto de las bases y realizar las conversiones entre estas, lo que nos debe de quedar muy claro es que cada lugar en el número tiene un valor que es diez veces mayor según te desplazas a la izquierda, y por lo tanto, diez veces menor cuando lo haces a la derecha. Este valor se multiplica por el número que está en ese lugar y todos finalmente sumados deben darte el número final.
Ahora vamos a comenzar con el sistema binario, es mucho más sencillo, pero puede parecer complicado, porque no es el que utilizamos en la vida cotidiana. Es el sistema de los ordenadores, y es vital conocerlo si queremos programar en Ensamblador.

SISTEMA BINARIO:

Este es el sistema utilizado por los ordenadores, y en general, por todos los aparatos digitales, debido a la importante característica de que su base es dos, lo cual quiere decir que sólo puede manejar dos dígitos distintos, que son el cero y el uno (¡Te suena!), y el menor número con el cual debemos abarcar el espacio más próximo a la izquierda es el 2.

El valor de las casillas en este sistema de numeración es muy fácil:



En el sistema binario para representar un número decimal, tan sólo deberemos activar las casillas correspondientes a la descomposición de dicho número en múltiplos de dos. 2, 4, 8, 16, etc.

```
|*|*|*|*|*|*|*|
 | | | | | | | \- Casilla de valor 1
 | | | | | | |
 | | | | | | \- Casilla de valor 2
 | | | | | \- Casilla de valor 4
 | | | | \- Casilla de valor 8
 | | | \- Casilla de valor 16
 | | \- Casilla de valor 32
 | \- Casilla de valor 64
 \- Casilla de valor 128
```

Este sistema si se presta atención no es difícil, por lo tanto, activa todas tus neuronas y ¡al grano!:
El primer número para analizar es el cero que EN TODOS LOS SITEMAS ES IGUAL—TODO A CERO—, eso quiere decir que si tomamos tres casillas para representar un número, estas estarán todas a cero.

```
|0|0|0| = 0
```

El número uno es casualmente el valor de la primera casilla, por lo que unicamente la activamos:

```
|0|0|1| = 1
      \- Casilla de valor 1
```

El número dos es también casualmente el valor de la segunda casilla, por lo que sólo se activará ésta:

```
|0|1|0| = 2 en decimal
   \- Casilla de valor 2
```

El primer problema se presenta cuando queremos representar el número tres, porque este número no tiene una casilla con su valor, ¿Qué hacer?, pues muy fácil, activamos la casilla de valor 1 y la de valor 2 que sumadas dan el valor 3:

```
|0|1|1| = 3 en decimal
   | \- Casilla de valor 1
    \- Casilla de valor 2
```

Para representar el número cuatro, como tiene una casilla con este valor, sólo nos limitaremos a activarla:

```
|1|0|0| = 4 en decimal
 \-Casilla de valor 4
```

Con el número cinco ¿qué hacemos?...
4+1 = 5:

```
|1|0|1| = 5 en decimal
 |    \- Casilla de valor 1
  \-Casilla de valor 4
```

El seis sería igual a 4+2 = 6:

```
|1|1|0| = 6 en decimal
 | \- Casilla de valor 2
  \- Casilla de valor 4
```

El siete, pues 4+2+1 = 7:

```
|1|1|1| = 7 en decimal
 | | \- Casilla de valor 1
 |  \- Casilla de valor 2
  \- Casilla de valor 4
```

Para representar el número 8 en binario necesitaremos la cuarta casilla, siendo imposible hacerlo con 3 (el mayor número representable en binario con tres casillas es el 7).
Hasta aqui bien, ya sabemos pasar un número decimal a binario, la inversa no tiene mayor dificultad, ¡intenta averiguar el siguiente número sin mirar!:

```
|1|0|0|1|1|1|0|1|
```

Pues muy sencillo, se suman los valores de las casillas activadas que son las de valor 1 + la de valor 4+ la de valor 8+ la de valor 16+ la de valor 128 :

```
1+4+8+16+128 = 157
```

Esto es lo mismo que lo que hacíamos de casilla - valor 1 por 1 más casilla - valor 4 por 1, (en el sistema decimal, recuerdas)...
Cuando tengamos que trabajar más en serio con el tema de las bases, te será de gran ayuda una calculadora para prácticar las conversiones y trabajar con ella cómodamente, si no la tienes y te la quieres comprar, asegurate que tenga bases BINARIA Y HEXADECIMAL (no todas las calculadores cientificas las tienen, la que yo utilizo es la HEWLETT PACKARD 20S).
En el próximo capítulo tenemos que ver muchas más cosas sobre el sistema binario. Como representar una palabra (16 bits) y una doble palabra (32 bits). Adiós.

Aprende a poner música en tus juegos

4ª PARTE

Music Instrument MIDI Digital Interface

Por:

José Manuel Ginés

El asesino entra en escena. La puerta del dormitorio chirría y sus pasos se oyen fuertes y estruendosos haciendo crujir la vieja madera del parqué. De repente una música tétrica hace que la sangre del jugador se hiele en sus venas. No hay duda. Los juegos de ordenador ganan mucho cuando la acción viene acompañada de una melodía. Por ello, es necesario que si queréis que vuestros juegos sean buenos y con calidad, tenéis que aprender como poner el toque musical adecuado. Continúa el estudio del interfaz MIDI y pronto sabrás como hacerlo.

En el siguiente artículo vamos a tratar dos interesantes programas diseñados para el Falcon, Apple y PC. Se trata de Notator Logic y Cubase Audio.

NOTATOR LOGIC

Empecemos por el primero, el Notator Logic. Este funciona a base de estructuras musicales tridimensionales. Cualquier combinación de secuencias, incluso procedentes de distintas pistas, pueden combinarse en una carpeta, y la propia carpeta puede contener sub-carpetas si es necesario, con todos estos objetos considerados como un todo integrado, de la misma forma en que un músico puede agrupar varios pensamientos e ideas.

Virtualmente no hay límite al número de carpetas que se pueden crear y el acceso a todas las funciones en cada nivel es fácil. Tan pronto como se empieza a usar Notator Logic, es posible crear de forma intuitiva una estructura jerarquizada cuyos componentes musicales se adaptan a la forma natural de pensar y trabajar del usuario. Las carpetas creadas pueden contener varias secuencias de una pista, así como arreglos multitímbricos de un coro completo, o incluso de una canción entera.

Dado que es el usuario quien estructura el programa, permanecerá siempre transparente y perfectamente adaptado a sus necesidades.

SIN LIMITES

Notator Logic acaba de una vez por todas con las barreras tradicionales gracias a su Sistema de Gestión Múltiple de Pistas, que permite el acceso a cualquier número de pistas y secuencias. Cada pista puede contener cualquier número de secuencias de cualquier longitud. Las superposiciones no suponen ningún problema.

Pasemos ahora a ver la SOFISTICADA TECNOLOGIA DE VENTANAS que posee. Cada ventana permite ampliación y reducción real, desde los pequeños detalles



Dos programas estrella

Notator Logic/Cubase Audio

microscópicos hasta la vista de pájaro de una composición de proporciones sinfónicas.

Cada ventana de Notator Logic tiene su propio conjunto de menús, una característica muy útil que muestra únicamente las funciones necesarias para la tarea que se está realizando.

Es posible abrir cualquier número de editores para una secuencia. Visualice y edite la misma grabación simultáneamente en los editores de Notas, Eventos y en el Hyper Editor. Como os podéis imaginar resulta imposible trabajar de forma más rápida e intuitiva. Todas las ventanas están interconectadas interactivamente y reflejan de forma inmediata cualquier alteración realizada en una de ellas.

Todas las configuraciones de ventanass definidas por el usuario son almacenables y accesibles mediante la pulsación de una tecla.

SECUENCIACION EXTRAORDINARIA.

El núcleo central de Notator Logic, su secuenciador, satisface las más altas exigencias. Una resolución máxima de nota de 1/3840 (960 pulsos por negra) garantiza la reproducción precisa de grabaciones en tiempo real a cualquier tiempo. El tiempo puede seleccionarse entre 0.5 y 9999.99 pulsos por minuto para adaptarse a cualquier tipo de tarea en el presente o en el futuro, como por ejemplo la sincronización de audio para vídeo o cine. La alteración de datos- permanente o no-destructiva- puede realizarse sutilmente a medida que la música está siendo reproducida, incluso mientras está sincronizada a SMPTE.

REPRESENTACION VIRTUAL DE SU SISTEMA MIDI.

Puede crear una copia virtual de su entorno de trabajo físico en la ventana de Entorno. Los componentes musicales "reales" tales como teclados, samplers y módulos de sonidos, son representados como objetos y pueden ser tratados exactamente de la misma forma que sus originales reales. Notator Logic permite interconectar los dispositivos MIDI o crear apuntes de pistas como se hace con magnetófonos analógicos. Este concepto lógico permite adaptar instantáneamente una composición a nuevas configuraciones del sistema MIDI. Ya no es necesario preparar las pistas antes de una grabación: basta con pulsar el ratón sobre uno de los instrumentos predefinidos, y la salida MIDI correcta se selecciona de forma automática e inmediata.

ACCESO A DETALLES MUSICALES

Notator Logic interconecta el secuenciador, el editor de partituras y los demás editores en tiempo real, y los datos grabados son visibles inmediatamente en todas las ventanas. Es posible seleccionar, insertar, desplazar o borrar notas y ver y escuchar simultáneamente los resultados.

Es posible alcanzar la máxima precisión gracias a los editores especiales: el editor de Eventos ofrece el máximo detalle alfanumérico, el versátil Hypereditor permite la edición de percusión o de curvas de controladores, y el editor Matrix visualiza la posición y el tono de las notas en una rejilla. Los tres editores son interactivos y pueden utilizarse simultáneamente.

MANIPULACION NO-DESTRUCTIVA DE DATOS.

Para preservar siempre las grabaciones originales, es posible trabajar con parámetros de reproducción reversibles. Resulta fácil transportar o añadir velocidad a secciones completas de la música.

EDICION MULTIPLE DE ESTRUCTURAS MUSICALES

Es posible realizar cambios rápidos y globales de parámetros para cualquier selección de objetos, sea cual sea su nivel. Además de las funciones estándar de Cortar, Copiar, Pegar y Borrar, se dispone de las herramientas y las funciones especialespara la tarea a realizar.

Dentro de este apartado vamos a ver tres puntos :

- Funciones de transporte versátiles. Controles dispuestos de forma adecuada combinan las funciones de transporte avanzadas del secuen-

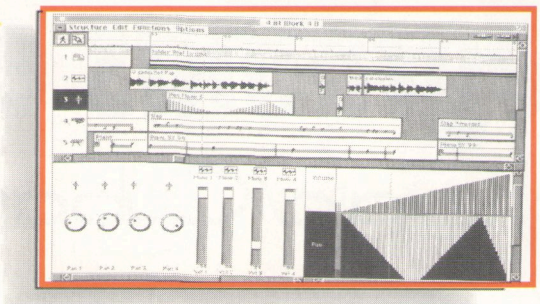
ciador con el interface simple de un grabador de cinta. Desplazando la Línea de Posición de Canción en modo de Arrastre, o pulsando con el ratón en la barra de compás, es posible acceder o saltar a la posición deseada de la canción.

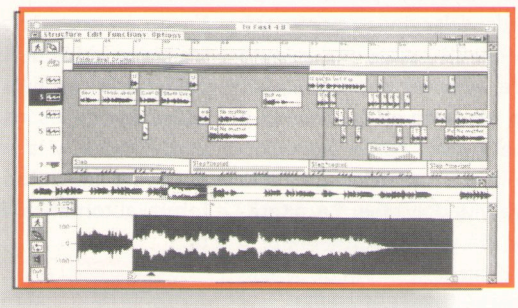
- Bucles encadenados. Cada secuencia o carpeta puede contener su bucle independiente de longitud variable. Es posible crear repeticiones complejas (bucles dentro de bucles dentro de bucles...) sin copiar nunca nada. De esta forma se ahorra tiempo y es posible editar pasajes musicales como si estuviesen en un sampler profesional.

- Sincronización con SMPTE y MTC. Notator Logic puede sincronizarse a grabadores de cinta o vídeo usando código SMPTE o MTC. Es incluso posible grabar o editar cambios de tiempo en tiempo real durante la sincronización y escuchar los resultados inmediatamente. Soporta también el estándar MMC (control de Máquinas MIDI). Otro apartado muy interesante que no debemos olvidar es el de LA NOTACION, y dentro de éste, para su mejor comprensión, analizaremos cinco puntos:

-Integración e interacción del sistema. Notator Logic es un programa de secuenciador y notación interactiva profesional todo en uno. Ofrece al usuario un entorno de trabajo accesible e intuitivo, desde la primera grabación MIDI hasta completar la maquetación de una partitura. Es posible imprimir una partitura en cualquier momento. Una característica única es la forma en que los niveles de visualización están interconectados en tiempo real: las ventanas de Arreglo y Entorno, los editores gráficos, el editor de Eventos y la pantalla de partitura.

- Facilidad de acceso. Las notas y una amplia variedad de otros símbolos musicales pueden ser insertados, borrados, transportados y desplazados usando el ratón o una de las prácticas herramientas contenidas en el Partbox, que se encuentra siempre a mano.





Dos pantallas del Notator Lógic

Editando la partitura. Pueden asignarse varios formatos de Estilo a cualquier selección de secuencias mediante la pulsación del ratón: la pantalla correspondiente de partitura se actualiza automáticamente. Pueden crearse pentagramas simples, dobles para piano con punto de visión definible, sistemas de pentagramas con polifonía multi-vocal y otras muchas opciones dentro de una partitura de cualquier tamaño. Aquí también, funciones reales de ampliación ofrecen distintos ángulos de visión de las secciones musicales seleccionadas. Es posible definir claves, cambios de clave, armaduras y canmbios de armadura, así como cuantización inteligente de la visualización tanto globalmente para la partitura completa como individualmente para cada pentagrama. Tanto el pentagrama de una pista, el extracto de la partitura de la sección de un instrumento o incluso la partitura completa de una composición pueden editarse simultáneamente en ventanas diferentes sin necesidad de complejas preparaciones previas.

- Polifonía. Es posible crear fácilmente cualquier número de voces polifónicas independientes para cada pentagrama. Es posible "explotar " todas las voces a pentagramas individuales, editarlas por separado y luego mezclarlas todas juntas de nuevo mediante una pulsación del ratón.

- Edición de página. Con Notator Logic, la visualización en pantalla es idéntica a la página maqueteada. Sea cual sea el factor de ampliación, es posible escalar la maqueta de forma precisa y editar notas de todas las formas mientras se monitorizan simultáneamente los resultados via MIDI. Opciones de formateo local y global permiten determinar libremente donde debe continuar un pentagrama en la siguiente línea y como deben ser los márgenes. Todo está optimizado para facilitar su uso.

CUBASE

A continuación vamos a hablar del otro programa para Falcon ya mencionado. Se trata, como ya hemos comentado del Cubase. Hablaremos de la tecnología del Cubase, pero para su mejor comprensión, vamos a dividir este estudio en varios apartados. Para comenzar...

EL CONCEPTO

Cubase es un concepto diseñado para contener las tecnologías más importantes de hoy y los avances del mañana.

Cubase considera todo esto en un formato musicalmente intuitivo, a tiempo real y gráfico. Puedes hacer lo que quieras, cuando quieras, sin perder la línea creativa.

El verdadero corazón del Cubase es la ventana de arreglos. Cada grabación es expuesta como un objeto gráfico. Esta puede ser cortada, copiada y pegada, o agrupada y trasladada, de cualquier manera mientras la música esté sonando. Tecleando siempre puedes tener una completa visión de tu canción.

LAS HERRAMIENTAS ADECUADAS

Diferentes trabajos necesitan distintas herramientas. Tienes la elección del método editor que prefieras, desde el control numérico pasando por el editor de piano-roll hasta el editor de la puntuación real , incluyendo sacar la partitura e imprimir. Cubase tiene las herramientas adecuadas para que puedas ser más productivo.

INTEGRACION

Los músicos del mañana contarán con todos los aspectos de la producción .¿Por qué esperar a mañana ? Cubase contiene la tecnología hoy.

Cubase provee un control remoto de tu grabadora usando el control del MIDI (MMC). Stop, Start, Cue, Record, todos los controles están integrados en Cubase.

Con un ADAT y un conector ACI, ADAT de Steinberg puedes controlar todas las funciones ADAT directamente desde el Cubase. Y no perderas una pista ADAT para el tiempo de sincronización.

Steinberg ha provisto al Cubase de un concepto a través de una grabación digital audio integrada y una serie de partituras dentro del Cubase. Y la innovación continua...

NOTACION MUSICAL

La notación musical, una de las más sofisticadas tecnologías, es una parte importante del Cubase. Su partitura, en realidad, al ejecutarse, hace que muchos programas permanezcan sólo dedicados a su trabajo.

El Cubase puede crear desde cero una partitura para una orquesta completa, todo en calidades profesionales de publicación.,

AUDIO DIGITAL

Audio digital, una tecnología que una vez fue sólo provechosa para unos pocos, es ahora una parte integral del diseño del Cubase. Audio es tratado en el mismo modo intuitivo gráficamente como las grabaciones MIDI. Puedes estar seguro que el Cubase Audio es la correcta elección para tí, porque Steinberg está encargado de permanecer en la cima de las nuevas manifestaciones hardware como ellos esperaban.

SENTIMIENTO

El corazón y el alma de una buena canción es el sentimiento. Secuenciadores y ordenadores software han tenido la reputación de no ser capaces, en absoluto, de capturar el ritmo natural y sentimientos de los músicos, el Cubase ha cambiado todo eso. Con el "quantizador Groove" de Steinberg, puedes crear los habituales ritmos templados que permiten que tu música suene como si fuese tocada en vivo, por músicos reales, no como tocada por una máquina. Eso es por lo que decimos que el Cubase puede hacer que tu ordenador tenga sentimientos.

La música basura también ha desarrollado una extensa bibliografía de sentimientos copiados. Simplemente carga estos sentimientos copiados dentro del Cubase e instantáneamente empieza a usarlos. Capturan el sentimiento de las canciones clásicas de los cincuenta a los días presentes y puedes incorporar su sentimiento en tu música con unos pocos movimientos del ratón.

LIBERTAD DE ELECCION

Cubase es provechoso para los ordenadores compatibles Atari, Macintosh e IBM-PC. Cualquiera que sea tu elección de ordenador, hay un Cubase para ti. Por supuesto, los ficheros Cubase son compatibles con los tres tipos de ordenador. Por ejemplo, puedes empezar tu trabajo con Cubase para Atari y más tarde cargar tus ficheros directamente dentro del Cubase para Windows. Tan sólo mantenlo en funcionamiento.

LA LINEA CUBASE

Hay un cubase para todos, desde el nivel inicial del Cubase, a través del Cubase o del MIDI integrado y la partitura Cubase imprimida hasta la Cubase Audio, todo en conjunto es integrado en la tecnología digital audio.

La política superior e innovadora de Steinberg te permite crecer con la tecnología Cubase. Puedes pasar de un programa a otro de cualquier modo, y puedes estar tranquilo, ya que, todos tus ficheros son compatibles entre sí.

LAS HERRAMIENTAS DE PRODUCCION DEL CUBASE

LA VENTANA DE ARREGLOS

La ventana de arreglos es a la vez un medio de grabación básico y un editor de canción gráfico.Tu música aparece claramente expuesta como objetos gráficos llamados partes.

Si quieres mover algo, tan sólo señala, presiona el botón del ratón, arrastra y suelta. ¡Movido! Para cortar una parte en dos, usa las tijeras. Para unir dos partes usa el tubo de pegamento. Si quieres borrar una parte, usa la goma de borrar. Encontrarás siempre las herramientas necesarias en la caja de herramientas, que aparece al presionar el botón derecho del ratón. Haz la prueba grabando algo.

La ventana de arreglos lleva el trabajo realizado fuera de la estructura de la canción. Sacar cosas fuera es fácil y rápido, puedes siempre anularlo si no te gusta. Crear nuevas pistas (por encima de 64 en cada una de las 16 ventanas de arreglo), elegir entre ellas, overdub, punch in o hacer grabaciones cíclicas en tiempo real.

No necesitas parar la música. Puedes orientar las líneas o patrones como deseas. Cubase es flexible y puede adaptarse a la forma en que quieras trabajar.Puedes ver siempre lo que estás haciendo. La ventana de arreglos tiene un zoom hacia delante y hacia atrás que funciona para una vista macro o microscópica.Las partes cambiadas aparecen en gris. La ventana de arreglos original de Steinberg ha sido frecuentemente copiada, pero nunca igualada. Es el centro de actividad en el Cubase y aquí es donde tus canciones, gráficamente toman forma ante tus ojos.

Pues bien, eso es todo por ahora. En el próximo artículo sobre el MIDI nos seguiremos centrando en estos dos mágnificos programas de creación musical. Más ideas, recursos, características y medios estarán a tu alcance.

J.M. Gines y Rocio Piñeiro

Microsoft© Macro Asso

Vamos a empezar a estudiar la materia interesante de esta sección, es decir, como se utiliza y como se le saca provecho a MASM 6.0.

Aunque, y como hemos explicado en artículos anteriores, nos centramos en uno de los últimos ensambladores de Microsoft, la materia que aquí analizamos es perfectamente aplicable a toda la serie de versiones de este mismo programa. Recordemos que el paquete 6.0 fue el primero que apareció como entorno integrado de programación, pero la versión 5.1 también puede valernos perfectamente, ya que las directivas y comandos que veremos más adelante, casi todas estaban ya implementados en ésta.

De todas formas, hasta que esto ocurra todavía queda un ratito. Por ahora, lo que vamos a ver vale para todas las versiones.

Pero empecemos ya. Acabamos el mes pasado viendo como podemos pedirle a MASM que nos fabrique una información especial que nos va a servir de gran ayuda a la hora de corregir nuestro programa, así como de ver su estructura interna. Hablamos de BROWSE. Pues bien, hemos llegado a un punto en las explicaciones, que para la perfecta comprensión de las mismas, necesitamos el actor principal de nuestra obra; un programa. Así que vamos a estudiar como se construye un programa en ensamblador y cuando lo tengamos empezaremos a ver como compilarlo y depurarlo con ayuda de todas las opciones de MASM.

EL PROGRAMA

Un programa es una estructura compuesta por órdenes en una sucesión lógica que nos permite llevar a cabo una función deseada. La estructura a la que hacemos mención se divide en bloques que a su vez se dividen en otros bloques en un subnivel inferior, y así sucesivamente. Estos bloques se denominan procedimientos. Un procedimiento se podría definir como aquella parte del programa que es capaz de realizar una función completa. Por ejemplo, si tuviésemos un programa que escribe en la pantalla "España es grande" se me ocurren, a bote pronto, dos procedimientos; el que borra la pantalla y el que escribe la secuencia de letras en la misma.

Por lo tanto, un procedimiento es un módulo dentro de un módulo mayor. La unión de estas partes hace posible que una función más grande se lleve a cabo. Como ejemplo de función más grande podemos tomar *una hoja de cálculo*. La finalidad de nuestro programa sería algo así como: "Permitir el cálculo rápido y automático de operaciones de todo tipo mediante los datos y fórmulas previamente introducidos en una plantilla de filas y columnas". De aquí, y a groso modo, se pueden intuir algunos procedimientos, como el de los cálculos matemáticos, el de control de la planilla, el que permite grabar y cargar datos a/desde el disco, el que controla la entrada de datos en las celdillas de las plantilla, etc.

Pero si os dais cuenta, todavía podemos ir más abajo en el nivel de nuestros procedimientos. Si tomamos el primero que hemos intuido, la de los cálculos matemáticos, nos damos cuenta de que habrá un procedimiento suma, otro resta, multiplicación, senos, tantos por ciento, etc.

Por lo que un procedimiento es una unidad funcional independiente que lleva a cabo una tarea particular dentro de un procedimiento mayor que puede ser el procedimiento principal, o función principal, u otro del que éste depende.

Pero veamos como se construye en lenguaje ensamblador. Por un lado, debemos indicar como comienza y en segundo donde finaliza. Figura 1.

Todo procedimiento ha de tener un nombre propio que nos ayudará en el futuro de dos formas. Primero, hará que el compilador sepa cuando estamos haciendo referencia a él, mediante una llamada o salto desde otra parte del programa. En segundo lugar, nos ayudará a nosotros a la hora de reconocer que función es la que realiza ese procedimiento. Por ello, especial cuidado hemos de tener a la hora de escoger el nombre propio de nuestro procedimiento. Por ejemplo, si tenemos un procedimiento encargado de borrar la pantalla, un nombre adecuado sería BORRAPANT, pero nunca BRPAN, o CLRSCR, o BRPT. Es decir, debemos tomar por norma que los nombres indiquen implícitamente la finalidad de esa parte del programa para que cuando nuestro proyecto sea grande, y tardemos mucho tiempo en concluirlo, nunca haya problemas a la hora de utilizar código que hemos desarrollado meses atrás. Utilizar nombres ingleses es también una buena costumbre. Primero, por que nos ayudará a estudiar los programas de Estados Unidos o Inglaterra (muy abundantes, por cierto) y en segundo lugar porque la idiosincrasia de este lenguaje permite utilizar nom-

Por:

Ramón López

La mejor herramienta para el programador en ensamblador es su compilador. En esta sección estamos analizando uno de los mejores del mercado. Aprende a manejarlo con nosotros.

emblER 6.0

bres que no por estar muy abreviados, dejan de tener significado. También, todo sea dicho, es el idioma por excelencia de la informática y la totalidad de los términos se acuñan en el lenguaje anglosajón. No es lo mismo etiquetar un procedimiento con el nombre MUEVEPANT a simplemente llamarlo SCROLL. Todos sabemos ya lo que significa. Por ello, utiliza nombres ingleses allí donde encajen mejor que los españoles y al contrario donde quieras. Total libertad en cuanto a esto, pero piensa siempre que has de hacer las cosas de forma que cualquiera entienda tu programa después.

Tras el nombre, vemos que hay una directiva ensamblador; PROC. Se trata de la abreviatura de procedimiento. Indica al compilador que en ese punto comienza una parte funcional del programa con lo cual le indicamos que a esta dirección de memoria pueden otras partes del programa hacer saltos.

Y, ¿cómo pueden ser estos saltos? Seguramente ya sabéis, y si no echarle un vistazo a el curso de ensamblador de este mes, que el microprocesador divide la memoria en segmentos. Los registros para tal fin actúan principalmente como punteros base de una zona de la memoria de 64 Kilobytes. Como hay 4 registros, la CPU puede direccionar (=trabajar con) tan sólo 256 Kilobytes simultáneamente. Esto, que os puede sonar a chino, es algo que dominaréis más adelante, por ahora os bastará saber que para la estructura interna de funcionamiento de nuestros programas, para el microprocesador el importantísimo saber si el salto o llamada se produce de un segmento a otro o dentro del mismo segmento. Figura 2.

A la primera modalidad se le conoce como salto largo y a la segunda como salto corto. Las operaciones que la CPU realiza en un caso u otro son diferentes por ello hemos de indicarle cuando nuestro procedimiento va ha ser llamado siempre desde el mismo segmento o cuando desde segmentos diferentes.

Para ello indicamos su TIPO en el comienzo de la declaración del procedimiento, mediante NEAR (cerca) o FAR (lejos). Esto ayudará al compilador a utilizar el código máquina correcto para RET y además indicará a la CPU cuantos datos debe empujar en la pila al llamar a la subrutina. No os asustéis, esto lo veremos más adelante en el curso de ensamblador.

El final de nuestro programa viene dado por dos datos principales. Por un lado el final que concierne a la CPU, es decir, la instrucción RET. Por otro lado el que tiene que ver con el compilador, la directiva ENDP (End Procedure, final de procedimiento).

Y ya está. Eso es todo. Por supuesto en el centro de esta estructura estarán nuestras instrucciones, que son las que en realidad hacen algo. Sin ellas, nuestro program no tiene funcionalidad alguna. Lo que hemos visto es más que nada unas reglas de cortesía que hemos de guardar para con nuestro compilador y sin las cuales jamásllegaremos a hacer un programa que funcione.

En el próximo artículo haremos y compilaremos un programa hecho por nosotros.

Figura 1. Procedimiento

```
Nombre  PROC  Tipo
        instrucción 1
        instrucción 2
        instrucción 3
        .
        .
        .
        instrucción n
        RET
Nombre  ENDP
```

Cuerpo del Procedimiento

Figura 2. Tipos de saltos



archivos .M

Un estándar de

Bien, el mes anterior ya estuvimos viendo cuales eran las características del sonido. También dejamos claro la importancia de estos en los videojuegos, ya que añaden una nota importantísima a la hora de hacer más latente la acción que se esta desarrollando en la pantalla. Por tanto, tenemos que aprender muy bien cuales son todos los formatos de que disponemos en la programación del PC para llevar a cabo una buena gestión del aspecto sonoro y musical de nuestros juegos.

Iniciamos la serie con los archivos MODULO abreviados, y más conocidos, como MOD.

DOS FORMATOS

Los dos formatos principales para la gestión de música en el PC son la síntesis y el muestreo. La tarjeta de audio Soundblaster, de Creative Labs, se ha estandarizado en el mercado como una de las opciones más claras a la hora de reforzar el PC, en cuanto al aspecto sonoro se refiere. Su programación, por lo tanto, puede ser enseñada sin miedo a las incompatibilidades, especialmente cuando ésta se fabricó con el antiguo estándar en mente; la ADLIB.

El hardware de la Soundblaster permite los dos tipos principales de creación sonora. Por un lado la síntesis, gracias a los chips FM que tiene implementados y que son programables para "sonar" de diferentes formas, o lo que es lo mismo, que nos hacen creer que en cada canal de sonido existe un instrumento diferente. Podemos, de esta forma, tener violines, trompetas, oboes, tambores, timbales, pianos, guitarras, etc.

Por otro lado, también se pueden muestrear sonidos previamente y hacerlos sonar en cuatro canales, como ocurría con un ordenador más preparado para estas lides: El Amiga.

Como podéis comprobar, se trata de una tarjeta bastante completa, lo cual ha hecho de ella la más vendida.

Pues bien, vamos a comenzar estudiando la segunda de las opciones, es decir el muestreo de sonidos y su posterior implementación en archivos especiales, así como su programación para que puedan ser utilizados en nuestros programas.

SAMPLERS

Un *sample* (muestra en inglés) es un archivo de grabación audio. Es decir, un conjunto de datos que, utilizados correctamente por programas especiales, permiten guardar digitalmente la información del sonido de un determinado instrumento.

Cogemos un micrófono y grabamos el sonido de nuestro violín, pues bien, eso es un sampler. La ventaja que estos tienen sobre la síntesis de sonido son tres principalmente:

1) Con la síntesis no tenemos tanto campo para la creación de sonidos fuera de lo normal. Tal vez, y sólo digo tal vez, un violín FM suene mejor que uno muestreado, al menos en una tarjeta de sonido no profesional, pero nunca podremos programar los registros de dichos chips para que suenen como un ladrido de un perro, una tos, o un martillazo.

2) No tenemos que pre-programar el sonido. Simplemente grabamos el sonido de la fuente audio que nos interese, por ejemplo, un compact disc, un casette, mediante un micrófono, etc.

3) La calidad real del sonido muestreado es inmensamente mayor que el sonido "robótico" del sintetizado.

Pero por otro lado, también tiene sus desventajas

1) Los archivos musicales con muestras ocupan mucho más espacio que los de música sintetizada.

2) La ejecución es más lenta, es decir, ocupan más tiempo de procesador, por lo que necesitaremos máquinas potentes (386 y 486)

3) La calidad estará siempre en relación a la frecuencia de muestreo. A mayor frecuencia, mayor tamaño de archivo y más grande será el tiempo que se nos come de proceso de la CPU.

En fin, cada uno deberá elegir su sistema en función de la calidad que quiera obtener. De todas formas, empezaremos, como ya he dicho, por el sistema de muestreo.

MOD

Un archivo MOD guarda toda la información necesaria para que pueda sonar, independientemente del sistema donde queremos hacerlo funcionar, por lo que pueden llegar a valernos incluso en aquellos sistemas que no tengan tarjeta de audio, ya que el altavoz interno del PC pude utilizarse como medio. Parece increible, ¿verdad?

Existen varios formatos del archivo MOD original, pero el que enseñamos hoy bien se puede tomar como estándar para el aprendizaje y más adelante veremos las diferentes versiones que han ido implementándose a través de los años.

LA CABECERA

En la cabecera de un MOD siempre encontraremos los siguientes campos (Ver tablas adjuntas):

1) En primer lugar el nombre de la canción. Si miráis cualquier archivo MOD mediante DEBUG, por ejemplo, podréis ir estudiando las diferentes partes de acuerdo con las explicaciones que os voy a ir dando. Para ello, situaros el el subdirectorio donde se encuentre el archivo y teclear:

DEBUG ARCHIVO.MOD

en formato MOD

calidad

Y después

D <enter>

El comando DUMP de DEBUG nos permitirá estudiar las entrañas del MOD.

Pues bien, como decía, los primeros 20 bytes del archivo contendrán el nombre la canción y los bytes que nos sobren estarán rellenos de ceros. Esta información es tomada por los ejecutantes MOD para introducir el nombre de la composición allí donde lo necesiten. Como sabéis las restricciones sobre los nombres de los archivos que nos impone el MSDOS nos impiden llamar una canción "El lago de los cisnes", sin embargo en este campo tenemos espacio para ello, siempre que no pasemos de 20 bytes, claro está.

2) También, y en la cabecera, a partir del byte número 20 (contando desde el principio del archivo y empezando en cero -recordar esto siempre-) tenemos el nombre del sample número uno. Tenemos en total 31 samples, es decir, en un MOD puede haber un máximo de 31 instrumentos o sonidos diferentes. El nombre, también en formato ASCII puro, ocupará no más de 22 bytes y lo que no sea utilizado se deberá rellenar de ceros. Ejemplo de nombres de samples pueden ser:

st-01:bass12
st-04:leader
guitarelectric

etc. El mes que viene continuaremos.

Por: Ramón López

El formato musical y sonoro .MOD se está convirtiendo en un estándar para la inclusión de música en los juegos actuales. Por ello, debes aprender cual es el formato de los datos que conforman este tipo de archivos.

Interioridades de un .MOD

Offset	longuitud	Descripción
0	20 bytes	Nombre de la canción. Los bytes no utilizados se rellenan con ceros.
20	22 bytes	Nombre del SAMPLE 1 Los bytes no utilizados se rellenan con ceros.
42	2 bytes	Longuitud del SAMPLE 1 El número indica el tamaño en palabras.
44	1 byte	FINETUNE. Sólo se usan los 4 bits inferiores.
45	1 byte	Volumen
46	2 bytes	Punto de repetición para el SAMPLE 1.
48	2 bytes	Longuitud de repetición del SAMPLE 1.

Los datos anteriores se repiten para cada uno de los 31 samplers.

Offset	longuitud	Descripción
50	30	Datos SAMPLE 2
80	30	Datos SAMPLE 3
110	30	Datos SAMPLE 4
140	30	Datos SAMPLE 5
.		
.		
.		
920	30	Datos SAMPLE 31

Offset	longuitud	Descripción
950	1 byte	Longuitud de la canción
951	1 byte	Byte = 127
952	128 bytes	Posiciones de la canción de 0-127 (datos 0-63)
1080	4 bytes	Letras "M.K."
1084	1024 bytes	Datos del patrón 00
.		
.		
.		
XXXX	1024 bytes	Datos del patrón N

La triquiñuela

CODIGO JUEGO

Por:

José L. Serrano Moreno

Más teclas

ESTADO_DEL_TECLADO(char &Puntero)

Este lector de Código Juego, preocupado por la utilización del teclado, nos ha enviado esta colaboración, que consiste en una librería en ensamblador para ser utilizada desde nuestros programas en C. Gracias.

Programa en Ensamblador para C.

Esta utilidad para C comprueba el teclado y coloca en Puntero[128] un 1 el Scan Code ha sido recibido y un 0 si el Break Code ha sido recibido.
Sólo hace falta instalarlo al principio del programa y desinstalarlo antes de salir del programa con

RESTAURAR_TECLADO();

Ejemplo de utilización de la librería

```
char Puntero[128];

main()
        {
        ESTADO_DEL_TECLADO(&Puntero);
// Instala drive del teclado
        ...
        ...
        if (Puntero[57]==1) printf("La barra
          espaciadora esta pulsada");
        else printf("La barra espaciadora no
                esta pulsada");
        if (Puntero[28]==0) printf("La tecla
           RETURN no esta pulsada");
        else printf("La tecla RETURN esta pul
                    sada");
        ...
        ...
       RESTAURAR_TECLADO();
// Desinstala el drive de teclado
        exit(0);
}
```

Listado 1. TECLADO.ASM

```
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT
VECTOR_TECLADO          DD      0
VECTOR_RELOJ            DD      0
DIR_TECLA               DD      0
TECLA                   DB      0

        PUBLIC  _ESTADO_DEL_TECLADO
ESTADO_DEL_TECLADO      PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        PUSH    BX
        PUSH    DI
        PUSH    DS
        PUSH    ES
        LES     DI,[BP+4]
        MOV     WORD PTR CS:DIR_TECLA, DI
        MOV     WORD PTR CS:DIR_TECLA+2,ES

        MOV     AH,35h
        MOV     AL,09h
        INT     21h
        MOV     WORD PTR CS:VECTOR_TECLADO,BX
        MOV     WORD PTR CS:VECTOR_TECLADO+2,ES
        MOV     DX,OFFSET CS:COMPROBAR_TECLADO
        MOV     AX,_TEXT
        MOV     DS,AX
        MOV     AL,09h
        MOV     AH,25h
        INT     21h

        POP     ES
        POP     DS
        POP     DI
        POP     BX
        POP     BP
        RET
_ESTADO_DEL_TECLADO     ENDP

        PUBLIC  _RESTAURAR_TECLADO
_RESTAURAR_TECLADO      PROC    NEAR
        PUSH    AX
        PUSH    DS
        PUSH    DX

        MOV     DX,WORD PTR CS:VECTOR_TECLADO
        MOV     AX,WORD PTR CS:VECTOR_TECLADO+2
        MOV     DS,AX
        MOV     AL,09h
        MOV     AH,25h
        INT     21h

        POP     DX
        POP     DS
        POP     AX
        RET
_RESTAURAR_TECLADO      ENDP
        PUBLIC  COMPROBAR_TECLADO
COMPROBAR_TECLADO       PROC    FAR
        PUSH    AX
        PUSH    DI
        PUSH    ES
        IN      AL,60h

        CMP     AL,0
        LES     DI,CS:DIR_TECLA

        JL      BREAKCODE

        MOV     AH,0
        ADD     DI,AX
        MOV     ES:[DI],1
        JMP     FIN

BREAKCODE:
        ADD     AL,128
        MOV     AH,0
        ADD     DI,AX
        MOV     ES:[DI],0

FIN:
        MOV     AL,20h
        OUT     20h,AL
        POP     ES
        POP     DI
        POP     AX
        IRET
COMPROBAR_TECLADO       ENDP

_TEXT   ENDS
        END
```

El contestador

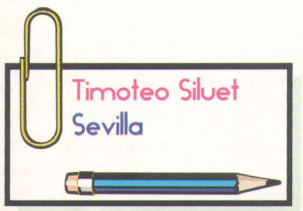


Todos los meses, en este apartado, queremos echarte una mano para que los problemas con tu ordenador no te quiten el sueño. Envíanos tus dudas a: Código Juego. Apdo 114. 11520 Rota (Cádiz)

Por:

Joan Esquerra

Solucionamos tus dudas



Otra forma de ver las cosas

Me llamo Timoteo, soy de Sevilla y me gusta a rabiar la informática y todo lo que está relacionado con ella. Empecé mi afición cuando se me ofreció la oportunidad de trabajar en una tienda del ramo y fue allí donde, a base de currar con ellos, se despertó en mí la afición por los ordenadores.

No me quiero extender mucho, porque me gusta mucho el "parloteo" y puedo comerme tres páginas de espacio antes de que os deis cuenta.

El caso es que me gusta la programación en lenguaje ensamblador y por supuesto, cómo no, crear videojuegos. Al ver vuestra revista me animé bastante, ya que la mayoría del la bibliografía existente es bastante reducida y difícil de entender. Todo ello hace que cuando quieras estudiar, acabes aburriéndote del todo.

Pues bien, a lo que voy. He oido hablar de algo así como el GRAPHIC MODE X, podríais darme más información acerca de esto, ya que creo que muchos programadores profesionales lo utilizan para hacer sus juegos.

CJ: Timoteo, tienes mucha razón en varias de las cosas que nos dices. En primer lugar te diré, saliendo a colación de lo que has dicho, que Código Juego tiene como primer finalidad la de enseñar a los que no tienen ni idea de programación en cualquiera de los lenguajes que aquí desglosamos mes a mes. Con ello no queremos decir que sea una revista para novatos, por lo menos el aprendiz sabrá encender un ordenador y teclar algunos comandos del MSDOS o del BASIC, así como tener una idea, aunque sea ligera, de como funciona un programa, ¿qué menos, verdad?

A partir de ahí, con esfuerzo e imaginación se puede llegar a hacer cualquier cosa gracias a esta revista.

El caso es que uno de los mayores problemas que nos encontramos a la hora de hacer grandes cosas con nuestro ordenador es que no existe documentación sobre los "pequeños" secretos de nuestro ordenador o que la que existe es en inglés o lo suficientemente confusa como para "aburrirse", según tus propias palabras. Ahí es donde nuestra -y vuestra- revista realiza su cometido con más fuerza, ya que nos vamos a dedicar a encontrar esa información que necesitas sobre tu tarjeta de vídeo, tu monitor, tu teclado, tu joystick, tu lenguaje, etc, y te la serviremos en bandeja, para que no te tengas que preocupar por buscarla, traducirla y finalmente, entenderla.

El modo 13X permite una visualización de 320x240 con cualquier tarjeta VGA estándar

Una vez hecho este inciso, pasemos a contestarte. Efectivamente, existe lo que se conoce como MODO GRAFICO X en la tarjeta de vídeo VGA. Como algunos sabréis, la tarjeta de vídeo trae un programa que la controla grabado en memorias ROM, para que siempre esté disponible y accesible (leer artículos sobre la tarjeta VGA de CJ1, 2 y 3). Una de las posibilidades que ofrece la ROM es cambiar de modo gráfico según la necesidad que se presente en nuestros programas. Por ello, tenemos modos alfanuméricos, o de texto y modos donde los gráficos son los reyes de nuestra pantalla. Para cada trabajo hay una visualización adecuada. Estos modos gráficos se conocen como estándar y se dispone de unos 20 aproximadamente, si se posee una tarjeta VGA. El problema llega cuando queremos encontrar un modo de vídeo adecuado para nuestro juegos, es decir, con el suficiente colorido y la máxima resolución posible, lo cual hará que nuestro programa esté dotado de una gran vistosidad. Sin embargo, la mayor resolución con la que contamos es de 300 pixels de largo por 200 pixels de alto. Casi nada.

Por ello, los programadores más expertos, inconformistas radicales, se han dedicado a darle vueltas a la tan usada tarjeta VGA para buscarle hasta el último de sus secretos.

El contestador



Gracias a que la tarjeta de vídeo es programable mediante lo que se conoce como "registros" ésta se puede acomodar a nuestras necesidades si sabemos reprogramarla a nuestro correctamente.

Ahí es donde entra en escena el modo X, también llamado Modo 13X, indicando que se trata de una variación sobre el modo 13h estándar. Una serie de programadores se han dedicado a tocar las tripas de la VGA para conseguir arañarle unos pixels de resolución más de lo normal, con lo que nuestro juego tendrá mas categoria visual debido a la mayor definición en pantalla.

El problema es que, por un lado, se requiere más código o programa, ya que hay que reprogramar la tarjeta, y por otro lado, es más difícil trabajar con la memoria de vídeo cuando utilizamos la tarjeta en esta modalidad.

Código Juego, en su sección de la tarjeta VGA, añadirá, llegado su momento, una serie de artículos que os enseñarán todos los secretos de este modo revolucionario.

Permaneced en antena.

Teresa Cervera
Barcelona

Música rebelde

Os escribo esta carta porque me he tropezado con uno de esos problemas de los que habláis en la revista, esos que impiden que puedas continuar con el proyecto que tienes entre manos. Por ello, me he decidido finalmente a enviaros este FAX.

Me encuentro en la actualidad realizando un juego de ordenador para PC y compatibles. Tras aproximadamente dos años de estudio del lenguaje ensamblador y experimentación en este campo realizando pequeños programitas, finalmente una idea, que considero brillante, cruzó mi mente, lo que hizo que me lanzase a la realización de un proyecto mayor. En el apartado de MOOKIE, "así se hace un videojuego para PC", de vuestra brillante revista he encontrado que Ramón López, director y programador de Código Juego, hace una buena explicación sobre el efecto FADE o de fundidos, que ya he estudiado e incluido en mi propio programa. Gracias a esto logré solucionar uno de mis problemas, pero no es por el que os escribo hoy.

En resumidas cuentas, lo que me preocupa en esta ocasión es que no se como poner la música dentro de mi juego y lo peor es que no encuentro información para poder llevarlo a cabo. ¿Podríais informarme sobre algún libro o fichero donde pueda conseguir información sobre este tema? No os hago todas las preguntas que tengo sobre este tema porque sería interminable. Os pido ayuda en general.

CJ: Nos pides ayuda en general, pero el tema es tan inmensamente amplio que no existen páginas suficientes en este número de CJ, y mucho menos en esta sección, para darte una explicación mínimamente detallada. Pero vamos a darte unas leves notas que esperemos te sean de utilidad.

En primer lugar te diremos que existen 3 formas fundamentales de añadir música o simplemente melodías más o menos complicadas a un juego de PC.

Los archivos de música .CMF han sido creados por CREATIVE LABS para la Soundblaster

La primera, y más utilizada, sería la música FM. Esta es la que se obtiene de la utilización de los chips generadores de sonidos que para tal fin estan incluidos en la tarjeta de audio. Por lo general se tratará de una Soundblaster o compatible con ella, por lo que los archivos musicales de este formato bien pueden ser los .CMF de Creative Labs o incluso .MID (estándar MIDI) si se sabe programar este tipo de archivos. Los diferentes instrumentos (trompeta, piano, violin, tambor, Hihat, etc...) se consiguen reprogramando los registros de los chips FM. Si utilizamos los archivos .CMF sólo deberemos cargar el driver SBFMDRV.COM que se incluye con el software de la Soundblaster y mediante las funciones que éste pone a nuestra disposición, él ya se encarga de esta reprogramación así como de arrancar y parar la música.

La segunda forma sería la utilización de archivos "módulo", más conocidos como .MOD. Para su correcta utilización existen multitud de fuentes de información que se pueden obtener en BBS o por via Internet, contactando con los múltiples grupos de programadores que hay a tu disposición. La información sonora sobre los diferentes instrumentos se guarda como "samplers" o muestreos de el sonido original los cuales se hacen sonar siguiendo un pattern o patrón que actua como un pentagrama. Son complicados de programar, pero su rendimiento es más alto, y la calidad sonora que se logra con ellos llega a límites insospechados, sobre todo si utilizamos ordenadores de rápida velocidad de proceso como los 386 y 486.

La tercera forma es la más fácil de hacer pero la que más memoria ocupa. Consistiría en crear la música en un instrumento aparte, o simplemente copiarla de alguna fuente externa (cassette, compact, etc.) y grabarla como un archivo .VOC. Después, mediante el uso del driver CT-VOICE.DRV, también incluido con el software de la Soundblaster, tan sólo hay que hacer sonar ese archivo de "voz". Esta forma no es recomendable en absoluto, especialmente si tenemos en cuenta que el disco se llenaría rápidamente debido al gran tamaño que estos archivos ocupan. Los archivos con este formato se suelen usar para efectos sonoros de corta duración, como explosiones, gritos, etc.

De las tres formas de poner música hablaremos en Código Juego, eso sí, en profundidad.

Una sección como esta es importantísima para los programadores como vosotros. En ella analizamos el mejor SHAREWARE Y DOMINIO PUBLICO del mercado para que sepáis como sacarle partido en vuestros programas

Para unos listados de alucine, aprende a utilizar este maravilloso controlador de impresión.

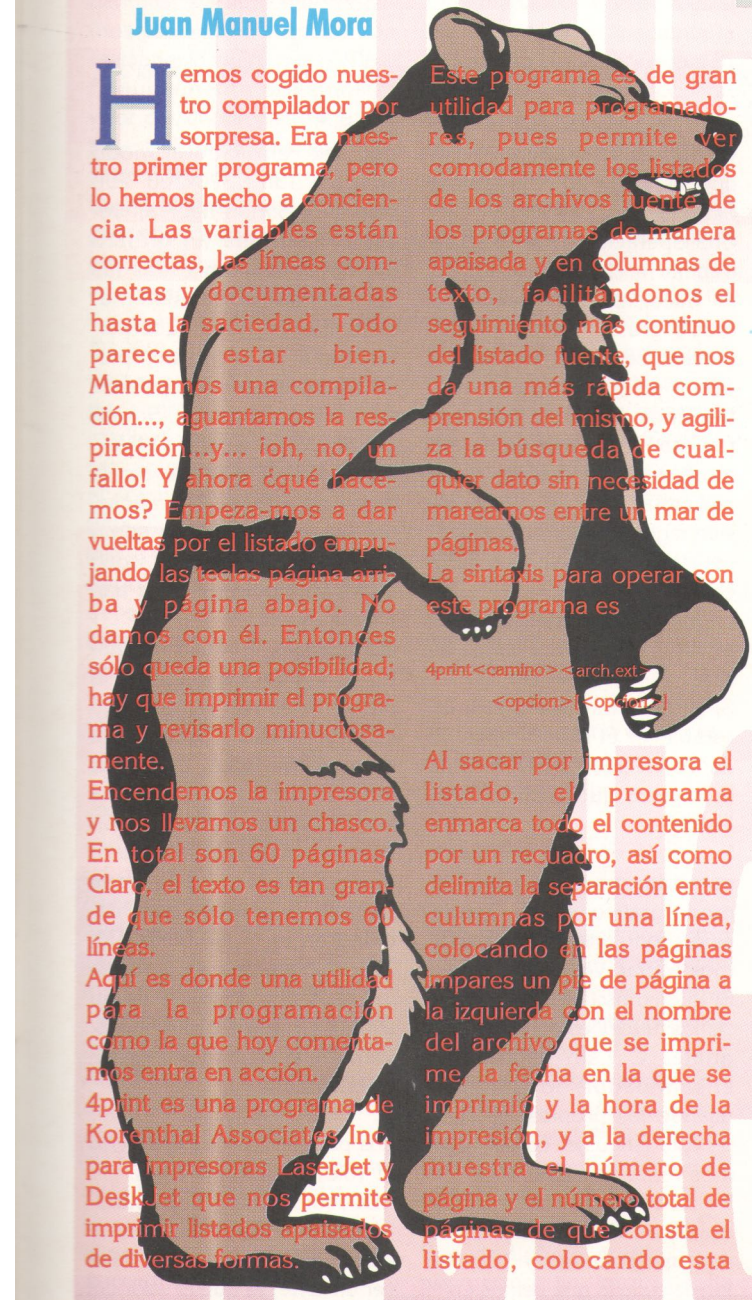
Un monstruo impresionante

Copyright Korenthal Associates, Tipo: Shareware

4print

Por:

Juan Manuel Mora



Hemos cogido nuestro compilador por sorpresa. Era nuestro primer programa, pero lo hemos hecho a conciencia. Las variables están correctas, las líneas completas y documentadas hasta la saciedad. Todo parece estar bien. Mandamos una compilación..., aguantamos la respiración...y... ¡oh, no, un fallo! Y ahora ¿qué hacemos? Empeza-mos a dar vueltas por el listado empujando las teclas página arriba y página abajo. No damos con él. Entonces sólo queda una posibilidad; hay que imprimir el programa y revisarlo minuciosamente.

Encendemos la impresora y nos llevamos un chasco. En total son 60 páginas. Claro, el texto es tan grande que sólo tenemos 60 líneas.

Aquí es donde una utilidad para la programación como la que hoy comentamos entra en acción.

4print es una programa de Korenthal Associates Inc. para impresoras LaserJet y DeskJet que nos permite imprimir listados apaisados de diversas formas.

Este programa es de gran utilidad para programadores, pues permite ver comodamente los listados de los archivos fuente de los programas de manera apaisada y en columnas de texto, facilitándonos el seguimiento mas continuo del listado fuente, que nos da una más rápida comprensión del mismo, y agiliza la búsqueda de cualquier dato sin necesidad de mareamos entre un mar de páginas.

La sintaxis para operar con este programa es

4print<camino><arch.ext> <opcion>[<opcion>]

Al sacar por impresora el listado, el programa enmarca todo el contenido por un recuadro, así como delimita la separación entre culumnas por una línea, colocando en las páginas impares un pie de página a la izquierda con el nombre del archivo que se imprime, la fecha en la que se imprimió y la hora de la impresión, y a la derecha muestra el número de página y el número total de páginas de que consta el listado, colocando esta misma información como encabezado en las páginas pares.

Entre sus numerosas prestaciones hemos de destacar que permite centrados de texto horizontales y verticales, centrados de texto horizontales o verticales, así como la modificación del tipo de letra estándar que el programa trae por defecto.

Entre sus opciones más destacables están:

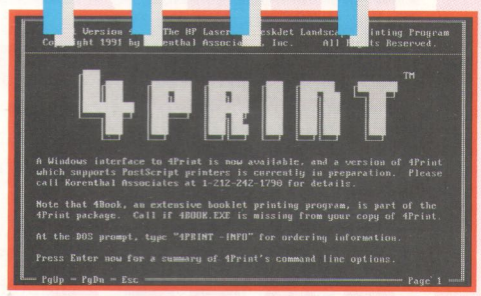
-1, permite la impresión de un archivo apaisado en una columna a 165 caracteres por línea.

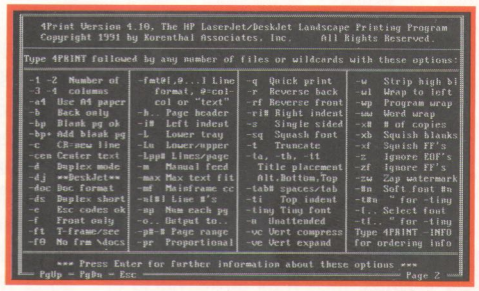
-2, imprime un archivo apaisado en dos columnas a 80 caracteres por línea.

-3, imprime un archivo apaisado en tres columnas a 55 caracteres por línea.

-4, imprime un archivo apaisado en cuatro columnas a 40 cara0cteres por línea.

-tiny, añadido a uno de las anteriores opciones permite el uso de la fuente tiny -



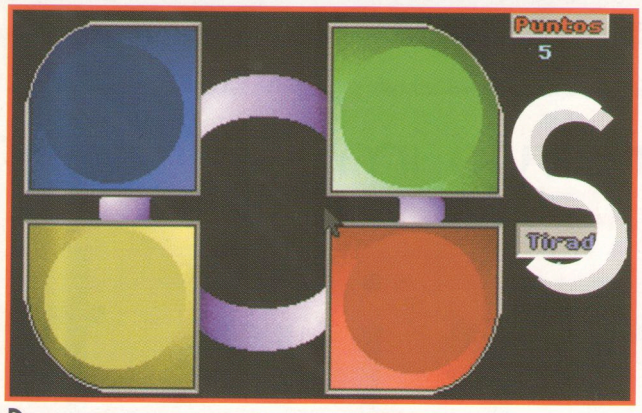


diminuta-, con lo que todavía podemos imprimir más páginas de texto en una sóla.

Para aquellos afortunados poseedores de una impresora DUPLEX, es decir que permite la impresión en las dos caras del papel a la vez, 4print también tiene entre sus opciones la posibilidad de aprovechar esta ventaja.

En definitiva una joya que aliviará muchos quebraderos de cabeza a todos los que programen también a aquellos, porque no, que les gusta ser ahorrativos, y es que el papel está muy caro.

Aprende a hacer un juego



Segunda entrega de nuestra serie de artículos

SIMON

Programación Gráfica

Por: Eduard Sánchez

Nota: El juego de SIMON fue incluido íntegro en el disquete de Código Juego número 2.

La evolución de las tarjetas de vídeo durante la historia del PC, ha sido realmente impresionante. Se empezó en el año 1981 con las tarjetas MDA y CGA. La tarjeta MDA sólo podía visualizar texto monocromo y la CGA (Color Graphics Adapter), como habréis podido comprobar en algunos juegos, sólo dispone de cuatro colores, los cuales resultan insuficientes para unas buenas pantallas gráficas.

Unos años más tarde, hacia el 1985, se lanzó al mercado la tarjeta EGA (Enhanced Graphics Adapter), la cual ya disponía de 16 colores en pantalla y era capaz de visualizar gráficos a una resolución máxima de 640x350. Esta tarjeta mejora notablemente a las dos anteriores, ya que la CGA permite una resolución máxima de 640x200.

Pero los usuarios informáticos aún no estaban satisfechos, por lo cual, un tiempo más tarde, apareció un nuevo modelo de CGA mejorada: la MCGA. La MCGA permitía un total de 256 colores simultáneos en la pantalla y disponía de una resolución máxima de 640x480, aunque con solo 16 colores. Imaginaros que cambio: en solo unos cuantos años se pasó de cuatro colores a 256 en pantalla!

El no va más apareció poco después con la VGA (Video Graphics Array). Esta tarjeta, la más utilizada en nuestros días, permite una resolución máxima de 640x480 con 16 colores y una calidad de visualización incomparable con las tarjetas anteriores. Mediante los modos VESA, y los nuevos modelos de VGA se pueden visualizar imágenes a altas resoluciones, que pueden llegar hasta 1024x768.

Actualmente se están lanzando al mercado nuevos modelos de VGA y nuevos tipos de tarjetas gráficas, que incluso pueden llegar a visualizar 16,'7 millones de colores... pero esto ya es otra historia!

Como habréis podido comprobar en la mayoría de los juegos actuales (aunque no todos), la resolución preferida es la 320x200 con 256 colores (MCGA), ya que en este modo gráfico es mucho más fácil trabajar, porque se dispone de 256 colores y una buena calidad de imagen.

Además, programar en 320x200 es mucho más fácil que programar en otros modos más altos: en el modo 320x200 256 colores cada pixel es representado mediante un 'byte', factor que permite un fácil acceso a la pantalla gráfica.

Vamos a adentrarnos un poco en la programación de la VGA, ya que es el siguiente paso que se realiza en el programa Simón.

Una operación esencial para trabajar a cien con la tarjeta de vídeo, es saber cambiar el modo gráfico de ésta. Este proceso se realiza con la función 0 de la interrupción 10h. Debemos colocar el valor 0 al registro 'AH' y el valor del modo gráfico a visualizar en el registro AL. Según el valor de este registro, podremos disponer de diferentes modos gráficos.

La tabla que se muestra en el listado 1, muestra algunos de ellos.

Si queremos, por ejemplo, establecer el modo 640x480 de 16 colores, lo haremos de la manera que explico en el listado 2.

En Simón se utiliza una rutina estándar para cambiar el modo gráfico, donde 'mode' es el modo gráfico que deseamos activar. Ver listado 3.

Si queremos únicamente utilizar las funciones puras del lenguaje 'C', sin recurrir al ensamblador, lo haremos usando los propios registros de éste lenguaje. como podemos ver en el listado 4.

Recordad que hay muchas maneras de acceder a los registros en lenguaje 'C', otra forma sería la que vemos en el listado 5.

Una vez cambiado el modo gráfico a 13h (320x200, 256 colores), Simón hace aparecer en pantalla un gráfico donde se indican todas las características del sistema, encendiendo unos cuadraditos, según se encuentren presentes las siguientes opciones en nuestro sistema: Sound Blaster, Ratón y Tarjeta VGA.

Pero antes de empezar con la carga de imágenes gráficas, me gustaría explicaros como acceder a la tarjeta de vídeo para poner pixels, realizar animaciones, etc...

Por ejemplo, la rutina siguiente, os ayudará a entender la programación en el modo 256 colores y del acceso a la memoria de vídeo. Este es el sistema clásico que se utiliza para poner un pixel de un color determinado en una posición determinada de la pantalla. Esta rutina solo funciona en el modo 320x200, 256 colores, que es el modo utilizado por Simón, para su estdio remitiros al listado 6. Vamos a analizar pasito a pasito esta rutina:

Los parámetros que enviamos al 'void' con los siguientes:
- La coordenada X del punto.
- La coordenada Y del punto.
- El color del punto. Que debe ser un 'byte' ya que el valor del

Nota: El programa SIMON fue incluido en el disquete de CJ2

pixel debe estar comprendido entre 0 y 255, factor que nos da un resultado de 256 colores en total.

Entonces accedemos al ensamblador y pasamos al registro AX el valor '0A000h' que es el valor del segmento de vídeo en el modo 320x200 256 colores. Todo 'byte' que almacenemos en esa posición de memoria, y de esta en adelante, nos aparecerá representado mediante un píxel en la pantalla, siempre que tengamos establecido el modo 13h mediante la interrupción 10h.

Seguidamente ponemos el valor de AX al registro ES, por lo tanto ES apunta al segmento de vídeo.

Luego tenemos que hacer un pequeño cálculo matemático para situar DI a la posición establecida por la variable X e Y, dentro del segmento '0A000'.

Recordad que las posiciones de memoria se expresan mediante el segmento y el offset; en el caso de esta rutina el segmento (ES) apunta a 0A000h y el offset (DI) apuntará a una zona de la pantalla determinada por DI. Para determinar el valor de DI deberemos hacer el siguiente cálculo:

(320 * Y)+X = posición relativa a la pantalla.

¿Por qué 320?, muy sencillo: cada línea en 320x200 tiene un máximo de 320 puntos. Si ponemos un valor superior a 320 entonces estamos apuntando a la siguiente linea. Mediante esa multiplicación estamos apuntando a la línea indicada por el valor 'Y'. Añadiendo X apuntaremos definitivamente a la posición correspondiente.

Todo este cálculo está almacenado en el registro 'AX' por lo cual lo deberemos traspasar al registro 'DI' mediante la instrucción 'MOV DI,AX'.

Finalmente ES:[DI] apuntan a una posición concreta del segmento de vídeo. Sólo deberemos almacenar un 'byte' en esa posición para visualizar un píxel: ponemos en el registro 'AL' el valor del pixel que deseemos, indicado por la variable 'color'. Entonces mediante la instrucción ' MOV ES:[DI], AL ' almacenamos ese píxel en la pantalla, lo cual provocará la visualización de un pequeño punto en la pantalla.

Otra manera de poner un píxel en la pantalla es mediante la utilización de la interrupción 10h (la destinada a las operaciones de vídeo). Los parámetros son los desarrollados en el listado número 7.

Su equivalente en lenguaje C sería la que se lista bajo el número 8.

Este sistema parece mucho más sencillo que el anterior, ya que así nos ahorramos mucha tarea de cálculos. Pero todo tiene ventajas y desventajas: poner un pixel mediante la interrupción 10h, funciona en todos los modos gráficos, mientras que mediante la anterior rutinilla en ensamblador, solo podemos poner un pixel en 320x200 256 colores. Pero la utilización de la interrupción es mucho más lenta que la otra y, en operaciones complicadas como en el caso de las animaciones, se aprecia mucho esa lentitud.

CARGAR FICHEROS GRAFICOS

Bien, pues, ya sabemos como acceder a la VGA y visualizar unos cuantos pixels... entremos pues en el mundo de los ficheros gráficos.

Simón no utiliza ningún formato gráfico 'famoso' como pueden ser el GIF, el PCX o el BMP, sino uno mucho más fácil de cargar: el 'Jovain VI'.

El formato 'Jovain VI' es un tipo de fichero gráfico muy poco conocido, pero lo suficiente como para ser soportado por algunos programas de tratamiento de imágenes. Por ejemplo, las imágenes de Simón fueron creadas en el 'Deluxe Paint II Enhanced' y grabadas a formato LBM (el formato del Deluxe Paint). Luego, mediante un buen programa, transformé el fichero LBM a formato 'Jovain VI'. Este tipo de fichero gráfico tiene la ventaja de que es muy fácil de cargar en cualquier lenguaje de programación, ya sea el C, el Pascal, el Ensamblador, etc.(ya lo veréis en el próximo número !).

Aunque los formatos PCX y GIF ocupan, posiblemente, cuatro veces menos, disponen de un sistema de compresión muy complicado, factor que provoca que el programador tenga que descomprimirlo y realizar un gran número de operaciones muy complicadas. En cambio 'Jovain VI' es un formato casi PURO. Esta está explicada en el listado 9.

Los primeros 16 bytes que forman la cabecera no nos interesan para nada, por lo tanto, en la rutina de carga, los saltaremos sin más. Los siguientes 768 bytes forman lo que se llama la paleta gráfica.

Cada color en el modo 256 colores (y en la mayoría de los otros modos) está formado por cierta cantidad de Rojo, Verde y Azul. Con diversas combinaciones de estos tres colores, podremos obtener una amplia paleta a nuestro gusto.

El color azul, por ejemplo, (color 1 de la paleta) dispone de los siguientes valores RGB:

R..... 0%
G.....0%
B..... 100%

El color rojo dispone de los siguientes:

R..... 100%
G..... 0%
B..... 0%

Haciendo combinaciones con estos tres valores podemos obtener miles de colores, pero desgraciadamente sólo podremos disponer de 256 colores simultáneos en la pantalla.

Como usamod utiliza el modo 256 colores, los 768 bytes de la paleta con resultado de multiplicar los 256 colores por 3, que nos da un resultado de 768 bytes.

Los siguientes 64000 bytes forman lo que se llama el 'bitmap' o el cuerpo de la imagen. Como no se utiliza ningún tipo de compresión en el fichero, entonces tenemos una imagen en pantalla de 320 pixels de anchura por 200 pixels de altura. En total 64000 bytes.

En total, el formato 'Jovain VI' en 320x200, 256 colores ocupa un total de 64784 bytes.

Al no tener que descomprimir, la rutina que carga los ficheros gráficos, realiza su tarea casi al instante, evitando así la espera por parte del usuario.

El proceso de carga de imágenes realizará tres pasos básicos:

* Saltar la cabecera
* Leer la paleta y definirla (mediante la interrupción 10h)
* Leer el cuerpo y traspasarlo al segmento de vídeo.

Al realizar estos tres pasos, obtendremos la imagen en pantalla de una forma rápida y sencilla.

Siento no tener más espacio para mostraros las rutinas para cargar imágenes, os prometo que aparecerán en el próximo número de CJ... No te lo pierdas!

Eduard Sànchez Palazón
Agosto'94.

SIMON El juego de los colores: Listados complementarios

LISTADO NUMERO 1

Modo (Hexa)	Modo	Tarjetas soportadas
04	320x200, 4 colores	CGA, EGA, MCGA, VGA
06	640x200, 2 colores	CGA, EGA, MCGA, VGA
0D	320x200, 16 colores	EGA, VGA
0E	640x200, 16 colores	EGA, VGA
0F	640x350, 2 colores	EGA, VGA
10	640x350, 16 colores	EGA, VGA
11	640x480, 2 colores	MCGA, VGA
12	640x480, 16 colores	VGA
13	320x200,256 colores	MCGA, VGA

LISTADO NUMERO 2

```
asm {
      xor ah,ah      // AH = 0, Número de la función.
      mov al,12h     // Modo 640x480 16 colores.
      int 10h
    }
```

LISTADO NUMERO 3

```
void set_video_mode(unsigned char mode)
{
        asm {
              mov ah,0
              mov al,mode
              int 10h
            }
}
```

LISTADO NUMERO 4

```
void set_video_mode(unsigned char mode)
{
    _AH = 0;
    _AL = mode;
    geninterrupt(0x10);
}
```

LISTADO NUMERO 5

```
void set_video_mode(unsigned char mode)
{
                union REGS registro;

                registro.h.ah = 0;
                registro.h.al = mode;
                int86(0x10, &registro, &registro);
}
```

LISTADO NUMERO 6

```
void pon_pixel(int x, int y, unsigned char color)
{
        asm {
              mov ax,0A000h
              mov es,ax
              mov ax,320d
              mul y            // Multiplica AX por 'Y'.

              add ax,x         // Añade 'X' a AX.
              mov di,ax
              mov al,color
              mov es:[di],al
            }
}
```

LISTADO NUMERO 7

Registro	Valor
AH (byte)	0Ch (0xC)
AL (byte)	Valor del pixel a visualizar.
BH	Página de vídeo (debe ser 0 en el modo 13h).
CX	Coordenada X
DX	Coordenada Y

Ejecutar interrupción 10h

LISTADO NUMERO 8

```
void poner_pixel(int x, int y, unsigned char c)
{
        asm {
              mov ah,0ch
              mov al,c
              xor bh,bh     // BH = 0
              mov cx,x
              mov dx,y
              int 10h
            }
}
```

LISTADO NUMERO 9

FORMATO JOVAIN VI:

1ª Parte....	16 Bytes de cabecera
2ª Parte....	768 Bytes de paleta
3ª Parte....	64000 Bytes de mapa de 'bytes' que forman la imagen gráfica.

CUANDO ESTES DESESPERADO...
CUANDO NO LO ENCUENTRES...
CUANDO TUS AMIGOS NO LO TIENEN...
CUANDO TE GUSTE TENERLO...

NO LO DUDES



CORRE A PEDIR TU SUPERDISCO

956-840246

El Superdisco de Código Juego cuesta tan sólo 950 pts. gastos de envio no incluidos.
Se trata de una colección de programas Shareware y de Dominio Público seleccionados para ayudaros a programar videojuegos.

Por:

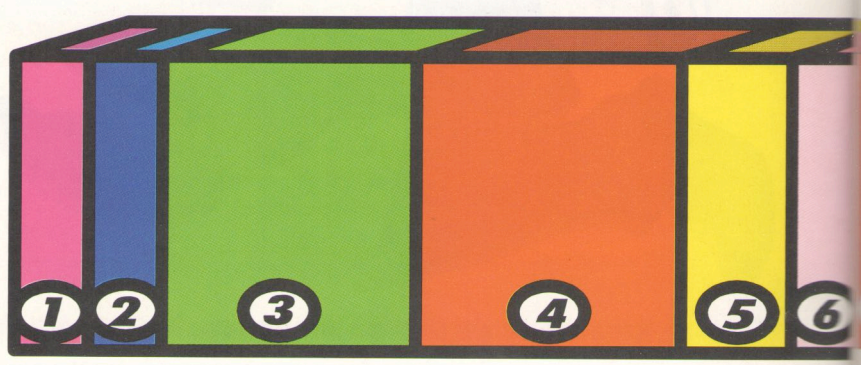
Ramón Lopez

EL MS DOS

y el programador de videojuegos

Seguro que más de una vez te has preguntado cómo se distribuyen los diferentes elementos de software dentro del ordenador y que memoria te queda finalmente para tus videojuegos. Unete a nosotros en este artículo y tendrás las ideas más, mucho más claras.





La intención de este curso es hacerte conocedor de los procedimientos a seguir cuando queremos aprovechar las funciones del MSDOS en nuestros juegos. Para ello, hasta ahora, hemos estado viendo cómo el sistema operativo era cargado desde el disco a la memoria. Esto nos ayudará a entender mejor cual es la estructura interna del DOS y cómo y por qué se requiere seguir un protocolo preestablecido a la hora de utilizar dichas funciones.

SITUACION DEL DOS

Nos quedamos en la tercera entrega de ésta sección justo cuando el sistema operativo reposa en el interior de la memoria RAM de nuestro PC. Es entonces cuando ya se ha realizado el pertinente testeo de los diferentes componentes que integran el ordenador a la par que todos estos han sido incluidos dentro del conjunto lógico de todo el sistema, pero, ¿cómo se sitúan los diferentes elementos del sistema operativo dentro de la memoria? En un sistema típico, o sea en un PC genérico o estándar, como prefiráis, el MSDOS no se mezcla o integra en un único bloque homogéneo sino que permanece separado incluso después de haber sido cargado desde el disco. Como seguramente acertaréis a comprender, diferentes ordenadores pueden tener diferentes configuraciones con respecto a la que aquí enseñamos. Esto se debe a que los archivos AUTOEXEC.BAT y CONFIG.SYS hacen que nuestro ordenador varíe en su formato interno. En el diagrama podéis echar un vistazo a lo que os comento, pero veamos, detenidamente, cada una de estas partes. Ello nos ayudará a conocer mejor cual es el espacio con el que contamos para nuestros juegos.

EL MODELO DE MEMORIA

Pasamos a comentar como se distribuyen estos elementos dentro del espacio direccionable del procesador en modo real, es decir los primeros 1024 kilobytes de memoria.

1) En primer lugar, la tabla de interrupciones del sistema operativo, que comienza justo en el primer byte de la memoria, es decir, 0000:0000. En ella se guardan los punteros de inicio de las rutinas de servicio del sistema operativo, el BIOS del sistema y todos los BIOS de todos los periféricos que están conectados al PC. El formato consiste en bloques de 4 bytes que indican en que dirección de la memoria se encuentra el comienzo de la rutina asociada a ese número en concreto. Estos cuatro bytes están guardados en de manera segmentada, es decir, segmento:desplazamiento y a la vez de manera invertida. Para que entendáis esto mejor utilizaremos un ejemplo. Supongamos que una interrupción comienza en la posición de memoria 1425:3456. Los dos primeros bytes, 1425h, indican el segmento de la memoria y los otros dos, 3456h, son el desplazamiento dentro del mismo. La forma en que nos los encontraremos en ésta tabla será:

56, 34, 25, 14

La dirección del segmento aparece después de la del desplazamiento y ambas se introducen en la memoria de forma invertida, o sea, byte bajo-byte alto. En el apartado del curso de ensamblador veremos el porqué en su momento. Por ahora simplemente aprender la manera caprichosa en la que se guardan estos datos y haceros saber que esto es una constante en el comportamiento de los microprocesadores Intel. La primera interrupción estará colocada en la posición 0000:0000 y la segunda 4 bytes después 0000:0004 y así sucesivamente. Para averiguar donde está el puntero de la interrupción 123 decimal simplemente la multiplicaremos por 4. las interrupciones ocupan justamente un kilobyte, o sea 1024 bytes. Si dividimos entre cuatro bytes que ocupa cada interrupción, averiguaremos que se pueden tener 256 interrupciones diferentes.

2) El segundo bloque viene ocupado por las tablas de variables del MSDOS. En ellas, el sistema operativo almacena información importantísima sobre el estado de PC, así como de las preferencias que hemos escogido para éste. Por ejemplo, el tipo de monitor que está conectado al PC, tipo de tarjeta de vídeo, información sobre la hora y fecha, teclado, etc. Recurriremos a estas tablas por dos motivos fundamentales. En primer lugar para informarnos sobre la situación de un determinado elemento de configuración o periférico. En segundo, cuando queramos cambiar algo para nuestro beneficio y que sabemos que el sistema operativo actuará en consecuencia. Son muy valiosas y más adelante daremos un completo listado de las mismas con sus respectivas descripciones y funciones.

3) El BIOS. Es decir, el que proviene del fichero IO.SYS. También, en lo que a IO.SYS se refiere, se engloban los dispositivos de control estándar (consola, auxiliar, impresora)



4) En cuarta posición el KERNEL del MSDOS, que proviene del fichero MSDOS.SYS. Se trata del bloque de rutinas de servicio de la interrupción 21h entre otras cosas. La palabra inglesa KERNEL se puede traducir por "meollo" o núcleo, es decir la parte más importante o funcional del sistema operativo.

5) Bloque configurable. Una vez que las secciones funcionales del sistema operativo ya han sido descritas entramos ahora en la parte que se configura con los archivos AUTOEXEC.BAT y CONFIG.SYS. Es decir, en esta zona de la memoria encontraremos probablemente los bloques de control de fichero (FCB) que se han abierto en CONFIG.SYS, así como se establecen en memoria los DEVICE DRIVERS o controladores de dispositivo, que como hemos aprendido son los anexos al BIOS que cada fabricante de periférico entrega con su producto para un perfecto control de ese dispositivo en concreto. Se añaden mediante órdenes DEVICE en CONFIG.SYS.

6) COMMAND.COM: La parte residente del controlador de órdenes o Commands en inglés se establece de forma permanente en esta zona de la memoria del PC, desde donde controlará nuestras entradas de teclado.

7) TSR. Los programas del tipo Terminate and Stay Resident, es decir, los más conocidos como programas residentes, se establecen en esta zona, en espera de la o las hot keys (teclas calientes) que los hagan despertar de su letargo. Por supuesto, en caso de que no existan, este área de la memoria se encontrará libre para la TPA.

8) TPA. Transitory Program área, o Area de Programas Transitorios. Es la zona donde nuestras aplicaciones (lo que vulgarmente conocemos como programas) entran en acción. Será, o deberá ser, la parte más amplia. Si utilizamos controladores de memoria extendida como HIMEM.SYS o QEMM386, se pueden llegar a tener cantidades de memoria disponibles muy cercanas a los 640k con lo que nuestras, cada vez más ambiciosas, utilidades informáticas tendrán todo el espacio que necesitan.

9) Memoria RAM libre. La zona de memoria que queda tras la última posición que ocupe nuestra aplicación hasta el límite de memoria RAM convencional (9000:FFFF) queda libre para que pueda coexistir cuantas aplicaciones quepan.

10) Parte transitoria de COMMAND.COM. La parte de COMMAND.COM que no sirve una vez que el programa de aplicación está funcionando se sitúa en la parte alta de la RAM de forma que puede ser "machacado" por los programas. Una vez que hacemos una salida al MSDOS, con lo que se finaliza el proceso con la aplicación en curso, se vuelve a cargar esta sección del interprete de órdenes. Si no se encontrase, el sistema operativo contestaría quejoso con un "No se encuentra COMMAND.COM, sistema detenido" y tendríamos que resetar el PC.

11) Memoria de vídeo (A000:0000). Dentro del espacio direccionable por el procesador (el primer megabyte de memoria) se le asignaron 128 kilobytes de memoria a la memoria de vídeo o VRAM. En un principio esta zona era más que suficiente para la cantidad de información gráfica que manejan nuestros programas normalmente. De todos es sabido que el PC ha crecido como la espuma y con él las aplicaciones y la cantidad de memoria de vídeo que estos necesitan, principalmente debido a la "gula" de los entornos operativos gráficos como WINDOWS. Ahora se recurre a complicados procesos de paginación de memoria para poder acceder a incluso 2 megabytes de VRAM, sobre todo en los adaptadores de vídeo SuperVGA.

12) ROM del Adaptador de Vídeo (C000:0000). Zona donde se instala el BIOS de vídeo incluido en la tarjeta del adaptador para que pueda ser direccionado por el procesador. Las funciones principales de vídeo están disponibles en esta dirección (int 10h)

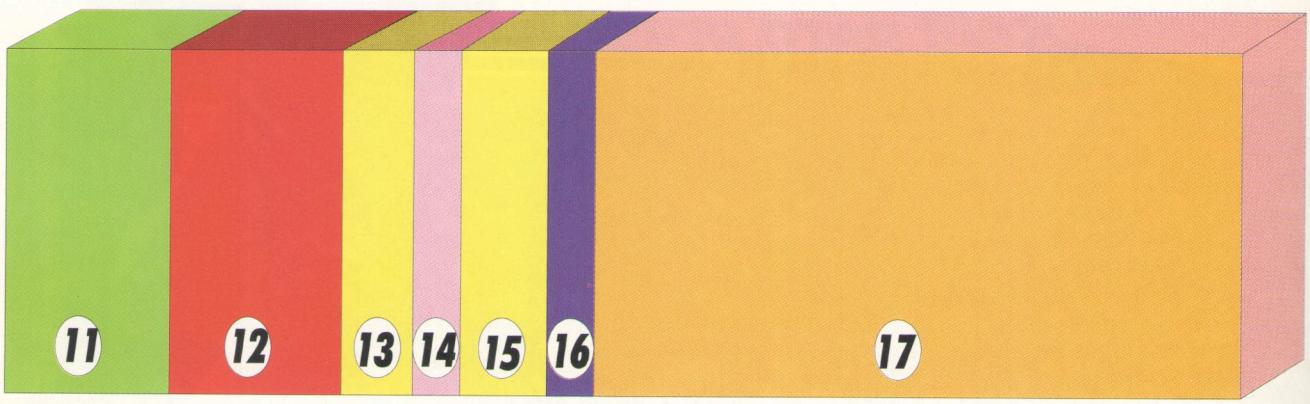
13) Tarjetas varias. La ROM de los adaptadores de otros dispositivos, tales como la controladora de disco o de redes locales se instalan en esta area.

14) Marco de Páginas. Zona donde el controlador de memoria Expandida LIM-EMS pagina la memoria de la placa correspondiente (hasta 32 megabytes). Es reubicable por lo que su dirección varía.

15) Memoria ROM BIOS. La BIOS general del sistema se encuentra siempre en la zona más alta de la memoria. Se incluyen aquí las rutinas básicas de entrada y salida.

16) HMA. La High Memory Area corresponde a unos 64 K que se pueden utilizar por el sistema gracias a unos manejos especiales que HIMEM.SYS lleva a cabo con la A20 (patilla con ese número en las líneas del bus de direcciones del procesador.)

17) Memoria extendida. La memoria que se "extiende" a partir del primer megabyte (1024k) de memoria convencional se llama memoria extendida, que no son direccionables por el procesador en modo real, es decir en el modo que funciona por defecto. Para ello, y siempre en modo real, debe utilizar un device driver especial conocido como HIMEM.SYS, el cual proporciona un interfaz seguro y compatible para acceder a páginas de memoria, las cuales siempre deberán ser conducidas hacia la memoria convencional para su uso por parte del microprocesador. Recordemos, sin embargo, que el sistema segmentado de memoria no funciona de esa forma en microprocesadores de la gama alta, como el 386 y 486, si estos se encuentran en modo protegido. En dicho modo, los procesadores mencionados pueden trabajar con el sistema de memoria plano, gracias a sus registros extendidos. La programación en modo protegido será motivo del curso de ensamblador mucho más adelante, cuando la programación en modo real sea manejada con suficiente soltura.



SUBSCRIBIRSE A

CODIGO JUEGO

ES UN BUEN BOCADO



Gracias a los descuentos especiales. Además, te aseguras de recibir en casa todos los meses la revista que más te gusta.

956-840246

HERRAMIENTAS DE PROGRAMACIÓN

DEBUGGERS

Depuradores del código

En capítulos anteriores hemos aprendido a utilizar algunos de los comandos de DEBUG. Hoy vamos a utilizarlos con un programa ejemplo para ponerlos en práctica.

Por:



Barón Rojo Bytes

Hemos visto ya bastantes cosas para estar lo suficientemente familiarizados con DEBUG y hacer nuestro primer programa en ensamblador.

Como he explicado en artículos anteriores, la calidad de DEBUG deja bastante que desear a la hora de ensamblar código fuente, debido a la forma en que éste trabaja. Como es natural en una herramienta pequeña, pero completa y potente en muchos aspectos, no ha sido implementada con gran cantidad de posibilidades como sus hermanos mayores (léase MASM y TASM). La sección de compilación con la que cuenta es diminuta y realiza todo el proceso en un sólo paso. Al contrario que MASM, por ejemplo, que compila en múltiples pasos, DEBUG no es capaz de resolver referencias externas, ni saltos a etiquetas. Por lo tanto, órdenes del tipo:

```
JMP SALIDA
CMP VALOR
CALL FAR SCROLL
```

...no están permitidas. Deberemos indicar exactamente a que lugar queremos saltar con la dirección absoluta o en todo caso (como cuando usamos JZ, JNZ, etc.) el valor con signo de el desplazamiento hacia adelante o hacia atrás. También, será necesario que nuestro programa contenga tan sólo un módulo, es decir, que todas las rutinas estén contenidas en el mismo programa que queremos compilar, ya que DEBUG no es capaz de buscar en librerías adjuntas ni de incluir archivos de código fuente. Nuestra aplicación deberá, pues, ser de "una pieza".

¿Para que sirve entonces DEBUG?, pues para hacer aplicaciones muy cortas, como pueden ser diminutos programas en formato .COM que nos ayuden con tareas muy simples. Por ejemplo, el programa que está listado en la figura adjunta es capaz de cambiar el modo de vídeo del 03h en el que el ordenador se sitúa por defecto al arrancar, al modo 13h de 320x200 pixels. Esta diminuta utilidad puede ser utilizada por un programa tipo .BAT siempre que queramos hacer que nuestro modo de visualización en pantalla cambie al anteriormente mencionado. Lo vamos a llamar VIDEO13.COM.

COMO CREAR CON DEBUG

El programa VIDEO13.COM es el colmo de la simpleza, pero nos puede ayudar muy bien a la hora de entender las dos formas de crear un programa con DEBUG.

1ª FORMA:

En números anteriores de Código Juego os he enseñado el comando A (de ASSEMBLE o ensamblar). Una vez introducida la orden...

```
A <enter>
o también:
A [dirección]<enter>
```

Un pequeño gran hombre
DEBUG.COM

No se trata de un depurador de grandes prestaciones, sin embargo nos ayudará a entender y apreciar herramientas más potentes. Todos lo tenemos en nuestro subdirectorio del DOS, y nunca nos querremos desprender de él.

Podemos empezar a insertar los comandos desde el indicador (-) de DEBUG directamente. Cuando llegemos al final tendremos que comprobar cuantos bytes ocupa nuestro programa y ordenar a DEBUG que los grabe de la siguiente forma:
Primero: Damos un nombre al archivo con el comando N (de Name, nombre)

-N VIDEO13.COM

Segundo: Insertamos en el registro CX el número de bytes que ocupa nuestro programa

-R CX
-CX:número<enter>

Tercero: Darle la orden de grabación en disco

-W

La orden W significa Write (escribir). Nuestro programa es guardado en disco con el nombre VIDEO13.COM, si lo ejecutáis ahora veréis que la pantalla y el cursor cambian de aspecto, así como las letras que ahora han aumentado de tamaño. Si queréis volver al modo normal de visualización (modo 03h 80x25, 16 colores) teclead: MODE CO80 y todo volverá a ser normal.
Este sistema está bien, pero no permite correcciones de ningún tipo una vez que el programa está guardado en disco. Por ello, la mayoría de los programadores utiliza la segunda forma.

2ª FORMA:
Consiste en crear un archivo de texto con las instrucciones que DEBUG deberá compilar más adelante, gracias a las posibilidades de redireccionamiento del MSDOS. Todos debemos de saber a estas alturas, y si no es así os recomiendo leer vuestro manual del MSDOS, que a un comando cualquiera podemos "pasarle" la información que está contenida dentro de un archivo y viceversa, es decir, podemos pasarle a un archivo el resultado de la operación de un determinado comando del MSDOS. Por ejemplo, la orden:

C:\>DIR >archivos.txt

...guardará en el archivo "archivos.txt" el resultado del hacer un DIR. Esto se conoce como redireccionamiento. Pues bien, la orden:

C:\>DEBUG <ordenes.txt

...pasará a DEBUG la información contenida en dicho archivo, que es la que podéis ver el la figura adjunta.
Como podéis comprobar, DEBUG toma por defecto que todos los números están en formato hexadecimal y por lo tanto no hace falta que indiquemos la "h" para el compilador.
De esta forma, si ha ocurrido un error, iremos al editor y efectuaremos la consiguiente corrección. Tras ello, volveremos a ordenar una compilación y veremos si el programa funciona.
Hasta el mes que viene, amigos.

Archivo ORDENES.TXT

```
N VIDEO13.COM   <ENTER>
A 100           <ENTER>
MOV AH,0        <ENTER>
MOV AL,13       <ENTER>
INT 10          <ENTER>
MOV AX,4C00     <ENTER>
INT 21          <ENTER>
                <ENTER>
RCX             <ENTER>
0B              <ENTER>
W               <ENTER>
Q               <ENTER>
```

Forma de compilar VIDEO13.COM

```
Desde el prompt del DOS

C:\>DEBUG <ORDENES.TXT
```

Gracias a las capacidades de redireccionamiento del MSDOS, podemos crear un archivo intermedio en formato texto que después actuará como línea de comandos de entrada automática para DEBUG.

En el error del mes pasado

Se nos coló un duende

Texto que no apareció en el artículo GRAFICOS 3D de la página 21 de Código Juego 3.



obteniendo así un par de coordenadas de pantalla, en los que pondremos el punto mediante la instrucción que use nuestro lenguaje para ello.

NOTAS SOBRE EL PROGRAMA DE EJEMPLO

El programa de ejemplo está hecho en TurboPascal. La razón de esto es que es un lenguaje bastante simple de comprender incluso por programadores noveles (con algunos conocimientos de Ingles). Es necesario comentar algunas lineas de este programa. El procedimiento principal, se llama PASA, y lo que hace es realizar las tres operaciones antes citadas de GIRO, TRASLACION y PROYECCION. Por lo tanto a esta función solo hay que pasarle la coordenada 3D (X,Y,Z) y nos devolverá el resultado final en las variables SX y SY. Si te fijas verás que en la primera línea del procedimiento convertimos X a negativo, esto es porque en el cuadrante que representa la pantalla está en la parte inferior del centro de coordenadas (el centro de coordenadas está en la esquina superior izquierda). Otro detalle en el que te habrás fijado, es que no se usan las funciones normales para el seno y el coseno que tiene el lenguaje. No las he usado porque son algo lentas debido a que el ordenador no se las maneja demasido bien con las operaciones en punto flotante. En su lugar, justo al comenzar creo una tabla con los senos y cosenos ya calculados, y sólo tengo que buscar en la tabla para conocer el resultado de la operación. Esto acelera considerablemente el cálculo. Para finalizar, ajusta la constante DELAYTIME para que el programa funcione a la velocidad adecuada para tú ordenador.

PARA FINALIZAR

Evidentemente, esto no es todo, sólo es el principio, pero vale como introducción. Aquí no hemos visto como se suprimen las líneas ocultas, ni como es el algoritmo de iluminación, ya que ellos arrastran una matemática mas compleja, de todas formas, con los conocimientos de este artículo, ya estás capacitado para intentar profundizar por tu cuenta a través de libros sobre gráficos.

Alberto García Serrano. Blit Boy

pasar a describir línea a línea lo que hace éste programa, vamos a ver una última opción del editor que seguramente es la que más vas a usar cuando estés programando. El TPascal tiene también una opción de ayuda integrada sensible al contexto. Todas éstas palabrejas significa que el TPascal es capaz de darte ayuda dependiendo de lo que estés haciendo en cada momento. Por ejemplo, sitúale sobre la orden WRITELN y pulsa CTRL+F1 y verás aparecer una ventana con ayuda e información sobre ésta instrucción.

COMO FUNCIONA EL PROGRAMA

Pasaremos ahora a ver lo que hace cada una de las líneas de nuestro programa. La primera línea, le dice al TPascal que lo que hemos escrito es un programa en Pascal. Tras la palabra reservada PROGRAM, aparece el nombre HOLA, este nombre es el identificador del programa, que puede ser un nombre cualquiera (sin usar caracteres extraños como la Ñ,ć,i,etc.. y empezando siempre por una letra). Después del identificador del programa, podemos ver entre paréntesis las palabras INPUT y OUTPUT separadas por una coma. Esto señala a TPascal que es un programa con capacidad de entrada y salida (E/S). No te preocupes si no entiendes ésto de la Entrada/Salida, ya que está parte es opcional en TPascal, y nosotros no la volveremos a usar, aunque tenla presente si usas otro compilador. De hecho toda esta línea es opcional en TPascal aunque nosotros la usaremos para ganar claridad. De ésta forma de ahora en adelante la primera línea la escribiremos como:

```
PROGRAM nombre_del_programa;
```

recuerda siempre terminar esta línea con punto y coma ";".

En la siguiente línea encontramos la palabra BEGIN, que significa inicio en inglés. Esta "instrucción" del TPascal, señala el inicio de un bloque de código, o sea, que a partir de ahí encontraremos el programa en sí. La línea que sigue es la clave del programa, ya que es la que escribe en la pantalla la frase

```
HOLA
MUNDO.
```

Write en inglés significa escribe y eso es exactamente lo que hace ésta . El formato básico de esta instrucción es:

```
WRITELN('texto a escribir');
```

Fíjate como terminamos la línea con ";". Siempre tendremos que acabar las líneas que contienen intrucciones de programa con ;. Un método nemotécnico para recordar que líneas deben terminar con ";" es ponerselas a las que realmente "hacen algo" en el programa, es decir, la instrucción BEGIN, aunque es obligatoria, no hace nada especial en el programa, sino que solamente indica el comienzo del mismo. Sin embargo, la instrucción WRITELN forma parte real del programa ya que es la que escribe la frase, y por lo tanto debe acabar en ";". La última instrucción del programa es END, que como seguramente sabrás significa FIN en inglés, y eso es precisamente lo que hace, señalar el final del programa. Observa que END va seguido de un punto, éste es como el punto y final del programa, y es obligatorio.

ESTRUCTURA DE UN PROGRAMA EN TPASCAL.

Entramos ahora en una sección algo teórica, pero no hace falta que te la estudies, simplemete haz una lectura comprensiva de ella y si algo no te queda muy claro, no te preocupes, lo verás mejor en los ejemplos prácticos (programas).

Un programa en TPascal está formado por tres partes bien diferenciadas, estas son:
la cabecera, la sección de declaraciones y el bloque de instrucciones (mira la figura 2). Pasaremos ahora a detallar cada una de estas partes.

.- cabecera. El formato general de la cabecera es:

```
PROGRAM nombre_programa (input,output);
```

nótese que la palabra PROGRAM no tiene por que estar en mayúsculas, ya que TPascal no distingue entre mayúsculas y minúsculas, por lo que podrás escribir los programas de la forma que más te guste. Como señalábamos antes, esta línea es enteramente opcional, sin embargo nosotros la usaremos, la única diferencia es que no usaremos (INPUT,OUTPUT), a no ser que no uses TPascal, en cuyo caso es conveniente ponerlo. La misión de ésta linea, es comunicarle a TPascal que estamos escribiendo un programa, y no cualquier otra cosa. No vamos a adelantar acontecimientos, pero te diré que TPascal soporta el uso de "programas externos" llamados unidades (UNIT), y que comienzan con una línea completamente distinta, pero eso ya lo veremos más adelante.

.- SECCION DE DECLARACIONES. Esta sección se descompone en varias subsecciones, son éstas:
Declaración de unidades
Declaración de etiquetas
Declaración de tipos
Declaración de constantes
Declaración de variables
Declaración de procedimientos
Declaración de funciones
No es necesario que estén todas en el programa, normalmente sólo necesitaremos una parte. Ahora daremos una breve descripción de cada una de éstas partes, pero antes aclarar que una declaración no es otra cosa que informar a TPascal que datos, funciones, u otro tipo de estructuras va a usar nuestro programa en Pascal. El primero de estos bloques, el de declaración de unidades, tiene la misión de informar al TPascal que unidades deben ser incluidas en el programa. Una unidad es algo así como una librería de procedimientos y/o funciones (instrucciones creadas por el programador) que están en un archivo aparte, consiguiéndose así lo que se llama programación modular. La definición de unidad, de función y procedimiento que he puesto, no es estrictamente correcta, pero nos valdrá por ahora. Hay un montón de unidades ya creadas en TPascal, como la unidad GRAPH, encargada del manejo de gráficos, o la unidad DOS, que contiene todas las operaciones relacionadas con el MS-DOS. Por ejemplo, si vas a usar instrucciones relacionadas con manejo de gráficos, deberás usar la unidad GRAPH. La sintaxis general de este bloque de declaraciones es....

Texto que no apareció en artículo Curso Pascal

Página 29, Código Juego 3

En el número de OCTUBRE tendremos muchas más cosas

CODIGO JUEGO

- Virguerías múltiples
- Curso ensamblador AMIGA
- Curso ensamblador ATARI
- Historia de los juegos
- Vuelve el curso WINDOWS
- Interfaz MIDI
- Ensambladores
- Depuradores
- Los sonidos del PC
- El MSDOS
- Hardware de AMIGA
- Lenguaje C
- Curso ensamblador PC
- El juego del mes
- Nuestro concurso
- Lenguaje Pascal
- La VGA, estudio a fondo
- Así se hace un juego MOOKIE
- Así se hace un juego en C



La revista número 5

Trae sorpresas

No te la pierdas..

¡¡¡ NOVEDAD !!!

Un libro de 300 páginas de aparición trimestral con todo lo necesario para programar como los profesionales

Todo tipo de información para programadores

Edición Limitada

PROGRAMACIÓN PRÁCTICA DE VIDEOJUEGOS

TRUCO LIST

Listados completos con explicación exaustiva

Por sólo 4000 pts

NÚMERO 1
Trimestre Oct-Nov-Dic 94

Condiciones de compra

Lee la tarjeta en la solapa de la revista.

Movimiento
SCROLL
SPRITES
VGA
MODO X
HARDWARE
FORMATOS GRÁFICOS
SOUNDBLASTER
INTERRUPCIONES
ADLIB
ANIMACIONES
SOFTWARE
GRATIS 2 DISQUETES
TABLAS DE REFERENCIA
PRESENTACIONES
TRUCOS PROFESIONALES



Los listados del libro están comentados por expertos para que puedas aprender con facilidad

Un precioso libro con 300 páginas lleno de contenido útil.

La más completa colección de listados seleccionados de todas partes del mundo

DOS DISQUETES: Con demos, listados y programas de aplicación para ayudarte a programar los juegos más increibles

En TRUCOLIST PC encontrarás toda la información que andas buscando y que o no encuentras o no entiendes. Se trata de un compendio de artículos sobre tu ordenador de un valor inestimable. Una verdadera biblia de la programación de juegos. En él obtendrás unos conocimientos teóricos y prácticos que no serás capaz de encontrar en ningún otro lugar. Trucos, explicaciones de expertos, notas técnicas, tablas de referencia sobre interrupciones, elementos hardware, drivers de memoria y de dispositivos, trucos usados en juegos, referencia sobre la Soundblaster, VGA, discos, impresora, ratón, joystick, etc. Corre y pídela ya porque la edición es limitada.

SOLO VENTA POR CORREO