La primera revista del mundo que enseña la programación de videojuegos

CODIGO JUEGO

975pts

Año 1, Número 3, Agosto 1994

Para todo tipo de ordenadores personales

Nuevo Curso de Pascal

Sección LA VIRGUERIA

Gráficos 3D

Aprende a hacer un juego hasta el final

MOOKIE

nuevo Amiga 500 Curso de ensamblador

Nuevo Curso Windows

8 424094 009756 00003

Ajatel S.L.

CODIGO JUEGO

Año 1, Número 3, Agosto 1994

Siempre pienso en ti. R.

Editorial: AJATEL S.L.

Director: Ramón López

Subdirectores: Joan Esquerra
Miguel A. Lobato

Directora de arte: Maite López Vidal

Colaboradores: Jose Manuel Ginés
Manuel Gallego
José A. Brihuega Parodi
Eduard Sánchez Palazón
Roberto Cruz
Alfonso Salgado Cachafeiro
María Jesús Cano Rueda
Juan Manuel Rial
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

Agosto 94

ESTE MES:

Ensamblador PC 7
Carta del lector 14
La tintilla 15
Así se hace un juego 16
El libro del mes 25
El juego del mes 26
La tarjeta VGA (III) 31
Lenguaje C 36
Ensamblador AMIGA 40
Ensamblador ATARI 42
Curso MIDI 44
Macroassembler 6.0 48
Curso Sonidos 50
La triquiñuela 52
El contestador 53
El buen soft 55
Así se hace un juego en C 56
El MSDOS 60
Los debuggers 64



NUEVOS CURSOS

Ampliamos la oferta de cursos dentro de Código Juego con Pascal y Programación bajo Windows

PASCAL Y PROGRAMACIÓN WINDOWS



La virguería

En la sección de la Virguería vas a ver las cosas que son capaces de hacer los programadores que leen nuestra revista. Animate tú también y envíanos tu programa.

20

Artículo

Gráficos 3D



Un nuevo colaborador de la revista, Albero García, nos ha mandado esta interesante aportación sobre el manejo de gráficos tridimensionales en Pascal. Aprende estos conocimientos para añadirlos en tus juegos

El buen soft

Qedit

Maravilloso editor de textos Shareware que nos sirve como entorno de programación.

Un juego divertídisimo que está hecho en C. Te enseñamos como programarlo paso a paso.

SIMON

El juego de los colores

Nuevo curso

Ensamblador AMIGA 500

Los amigos del Amiga ya tienen sitio en CJ con este curso.

Editorial

CODIGO JUEGO

Director de CJ

Ramón López

La realidad que se acerca

A los que nos gustan los juegos de ordenador se nos cae la baba cuando echamos la vista al frente y empezamos a ver lo que se nos viene encima. En pocos años nos vemos sentados ante supermáquinas de juegos, ya sean ordenadores o consolas que harán que nos introduzcamos de lleno en mundos imaginarios de los que tan sólo nos faltará el olor y el tacto, puesto que el resto de nuestros sentidos estarán plenamente satisfechos con miles de estímulos reales. Estímulos que tornarán los de la vida diaria y rutinaria en aburridos, convirtiendo esos mundos virtuales llenos de colorido, sonidos y acción en lugares donde nos gustaría vivir eternamente, puesto que en ellos somos los reyes, los príncipes valientes, los conductores de coches más veloces, los pilotos intrépidos, los soldados de la libertad, los guerreros medievales, etc...

La importancia, por tanto, que cobraremos los creadores de software será muy relevante. Lo que ahora es programación lúdica, y por ello tachada de falta de categoría profesional así como de estar orientada hacia un mercado amplio pero poco exigente, llegará a ser la que domine este nuevo campo que se acerca a pasos agigantados y firmes por el horizonte informático.

Los que programamos juegos para ordenador no tenemos más remedio que conocer nuestra máquina muy a fondo. Esto, por un lado, es una desventaja, ya que tenemos que estudiar mucho más que cualquier programador de alto nivel que cómodamente utilizará las librerías, previamente concebidas por programadores de lenguajes de bajo nivel. Pero, por otro lado, estos conocimientos nos convierten en los que mejor pueden llegar a programar nuestros ordenadores. Somos los druidas que saben mezclar la pócima en sus cantidades exactas y muy pocos más son capaces de lograrlo.

Sin embargo, una cosa esta clara; llegar a ser un buen programador de juegos exige dos cosas primordiales de parte de la persona que pretende llegar a serlo. Por un lado, ser estudioso. No me refiero a ser un empollón hormiguita, ratón de biblioteca, sino a ser conscientes de que esto puede llegar a ser una profesión que nos llege a satisfacer más que ninguna otra del mundo, reportándonos dinero y orgullo por lo conseguido. Deberemos, pues, otorgarle la suficiente seriedad. Por tanto, se le deberá reservar un espacio diario al aprendizaje de los temas que competen al programador, así como a la investigación y lectura de las nuevas tendencias para mantenernos al día en un mar tan cambiante como es el de la informática.

En segundo lugar, el programador deberá ser un experimentador nato. Es decir, no tendrá nunca miedo a realizar sus propios programas por muy malos que sean en un principio. Esto es vital, ya que el ordenador, al ser tan complicado en un principio, no nos abrirá sus puertas de par en par. Seremos nosotros los que tengamos que hacerlo poco a poco mediante la mejor técnica para ello: El ensayo-error.

Tened pues en cuenta estos dos consejos. Séd constantes y pronto veréis los frutos de vuestro trabajo.

A lo anterior debemos añadir que para llegar a ser un buen programador necesitaremos un buen apoyo. Libros y revistas principalmente. Ahí es donde entramos nosotros y el resto de las revistas de informática. A parte de estos , todo programador tendrá que, especialmente en el caso del lenguaje ensamblador, estudiar los programas que otros crearon y de ahí sacar sus conclusiones. Yo, particularmente, ya he perdido la cuenta de los que he desensamblado con el sano fin de estudiarlos y aprender

--*.

En lo que a la revista de este mes se refiere hay bastantes cosas que destacar. En primer lugar añadimos un nuevo curso para satisfacer las peticiones recibidas al respecto. Se trata del lenguaje Pascal, que también puede usarse perfectamente como interfaz como otros lenguajes como el lenguaje ensamblador, estrella indiscutible de nuestra revista. Y si no, no tenéis más que ver el artículo sobre IGOR:Objetivo Uhikokahonia del número anterior. Este magnífico programa ha sido creado tanto en ensamblador como en Pascal.

La potencia y sencillez de aprendizaje de este lenguaje lo hace ideal para comprender la estructura de funcionamiento de otros más complejos, convirtiéndolo en una buena plataforma de trampolín hacia el mundo de la programación. Tras valorar esto, la redacción de la revista ha decidido dedicarle un espacio considerable en este número y en los venideros. Los lectores de CJ están de suerte, no sólo por lo dicho, sino porque también empezamos a estudiar la programación bajo entorno Windows. Este entorno operativo se ha adueñado de las aplicaciones serias del PC, pero no por ello deja de ofrecer posibilidades a los juegos, algunos de los cuales podemos ver ya funcionando dentro de sus conocidas ventanas. Para que sepáis como crear vuestros juegos para Windows, así como para ganar unos conocimientos que se me antojan imprescindibles en la programación de cualquier otro tipo de software para este entorno, iniciamos en CJ3 un curso que, al igual que los otros, entra a forma parte de nuestro contenido habitual.

En último lugar, pero no por ello desdeñables, mencionamos los cursos que ya tenemos y los artículos que continúan. Los cursos de Ensamblador, C y MIDI empiezan a tener consistencia. En ensamblador hacemos nuestro primer programa, que aunque simple, comienza a daros una idea sobre la estructura o plantilla que usaremos en nuestros programas. El lenguaje C continúa con el estudio de las importantísimas funciones. El MIDI sigue su enseñanza de como conectar el ordenador y nuestro instrumento musical preferido.

Tenemos nueva sección, la cual considerábamos imprescindible para los programadores, que sin llegar a ser expertos, sí tienen ya algunos conocimientos sobre los lenguajes que aquí enseñamos. Se trata de la VIRGUERIA, una serie de artículos de nuestros colaboradores en los que veremos como se hacen muchos de los trucos y fantásticas animaciones que se usan en los juegos profesionales. En este número, nuestro colaborador en la materia, nos enseña como hacer una animación en ensamblador que se puede utilizar perfectamente en nuestros juegos.

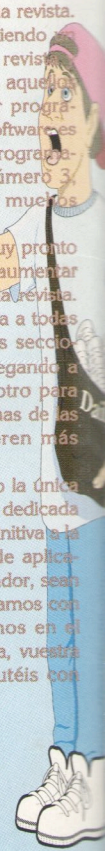
Un artículo más, concretamente el de gráficos tridimensionales, nos sumerge en este tipo de programación hecha en Pascal.

Este mes no llevamos disquete y francamente lo sentimos, pero nuestra revista es todavía pequeña y no siempre podemos hacer frente a este gasto, pero no os preocupéis que esto no será siempre así. Los que os hayáis subscrito a Superdisco o los que en su defecto lo compréis este mes, podréis recibir los programas de los que se hablan en este número y que no tenéis ya de los dos discos anteriores que os entregamos en la revista.

El superdisco sigue pues siendo un apoyo suplementario a la revista especialmente orientado a aquellos que no pueden conseguir programas o que no saben que software es el más indicado para la programación. En el superdisco número 3 podéis encontrar estos y muchos más.

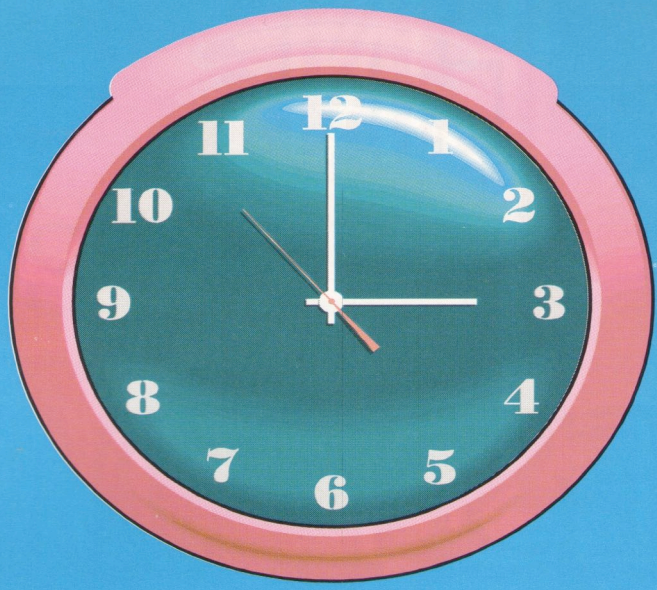
Finalmente deciros que muy pronto nos veremos obligados a aumentar el número de páginas de la revista. Por un lado para dar cabida a todas la peticiones sobre nuevas secciones y artículos que van llegando a nuestra redacción y por otro para aumentar en tamaño algunas de las ya existentes que requieren más espacio.

Código Juego sigue siendo la única revista que esta totalmente dedicada a la programación y en definitiva a la creación en mayúsculas de aplicaciones para nuestro ordenador, sean del tipo que sean. Continuamos con la promesa que os hicimos en el número 1, de mejorar esta, vuestra revista. Espero que disfrutéis con este número.



Ajatel S.L.

CODIGO JUEGO



Ya sólo quedan 3 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: El PC de IBM vió la luz en la década de los ochenta, pero ¿en qué año?

Cuando lo sepas, toma la última cifra de dicho año realiza las siguientes operaciones:

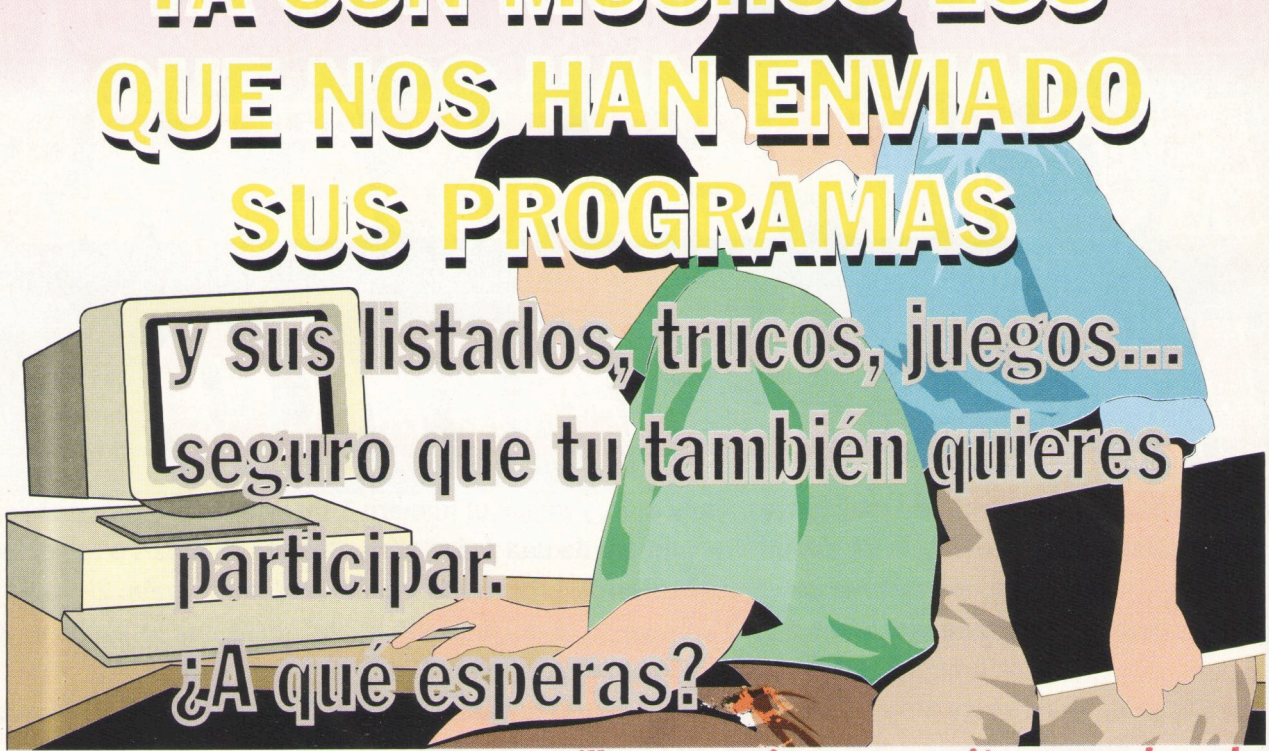
Toma las dos cifras anteriores (Preguntas de Código Juego 1 y 2) y súmalas a la de este mes. El resultado se lo vuelves a sumar al código ASCII de la letra a minúscula. Obtendrás un número de 3 cifras que debes apuntar porque será con el que jugemos el mes próximo. Hasta entonces.

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.

CODIGO JUEGO

Curso de lenguaje Ensamblador para PC

3a parte

Profesor

Ramón López

El pasado mes vimos, por encima, como trabaja el microprocesador a la hora de ejecutar una orden cualquiera. También aprendimos que para que un comando pueda ser llevado a cabo por rey de los "chips" es necesario el transcurso de unos componentes muy especiales de éstos; los registros, almacenamientos temporales de información muy usados por el microprocesador para guardar los resultados de las operaciones realizadas con anterioridad.

LOS REGISTROS DE DATOS

De los registros que vimos el mes pasado, que eran todos aquellos de los que dispone el microprocesador, vamos a comenzar con el estudio de los registros de datos.

Como su nombre indica, nos permiten trabajar con los datos con los que se realizan las operaciones dentro del ordenador. Por ello, como os podréis imaginar son los que más se usan, pero no os preocupéis, los registros no se rompen o se desgastan con su uso. Podéis utilizarlos cuantas veces queráis y siempre estarán allí para volver a ser usados.

Estos registros son sólo cuatro. Se les denomina alfabéticamente A, B, C y

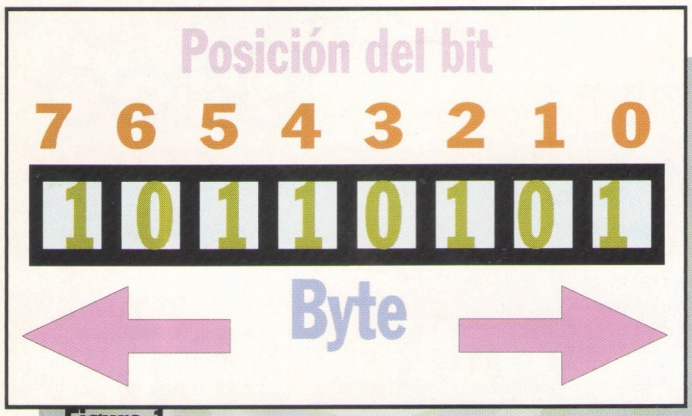


Figura 1

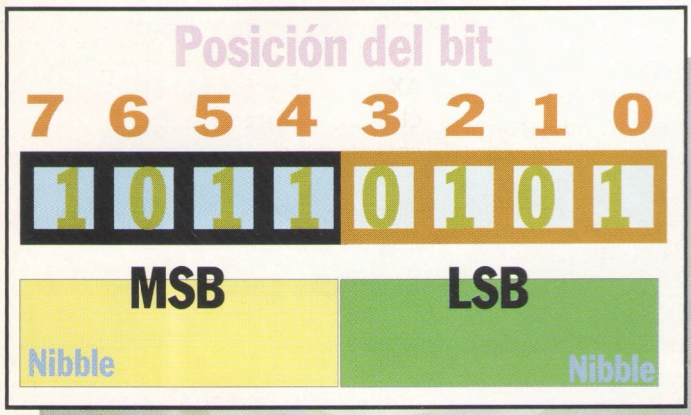


Figura 2

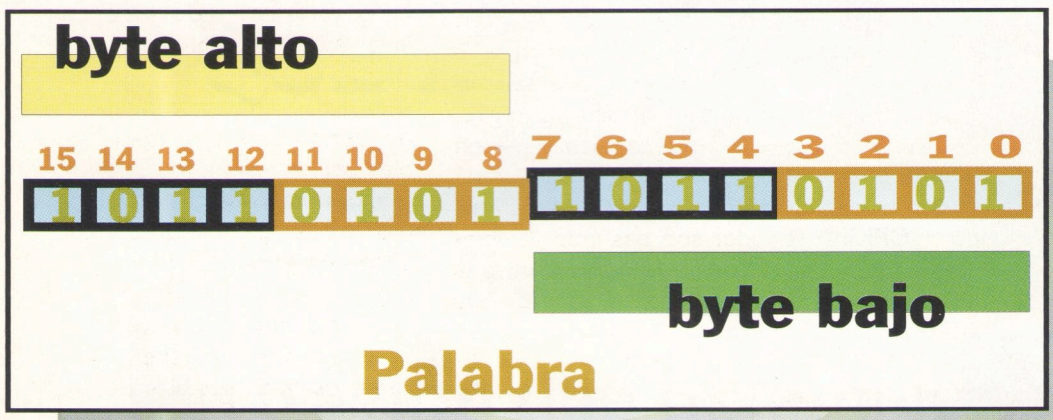


Figura 3

LAS PALABRAS DE 16 BITS Y LAS DOBLES PALABRAS DE 32 BITS SON EL REALIDAD 2 Y 4 BYTES. SE UTILIZAN CUANDO NECESITAMOS HACER OPERACIONES CON NUMEROS SUPERIORES A 255.

D. ¿Por qué sólo cuatro?, la verdad es que no hacen falta más si se llega a programar con cierta prudencia en el uso de los mismos. De vez en cuando se echa de menos algún otro, pero como veremos más adelante, no tenemos de qué preocuparnos.

Ya sabemos el nombre de ellos, pasemos ahora a los apellidos que es siempre una X. Esta indica que el registro es de 16 bits en lugar de 8 bits, pero vamos a profundizar en esto un poco más.

Como ya sabemos de artículos anteriores, los microprocesadores necesitan de los BITS (unos y ceros) para entenderse con el mundo exterior. Estos, no son sino impulsos eléctricos que sirven como los golpes sonoros de un tamtam para que la CPU actúe de la manera que nosotros queremos. También hemos visto que los bits se agrupan en cadenas de ocho de ellos para poder ser tratados en unidades más grandes llamadas bytes. Un byte son 8 bits agrupados de la forma que podéis ver en la figura 1.

Como casi todo en informática, la numeración de orden comienza siempre en cero en vez de a partir de uno. De esta forma tenemos el bit cero, bit uno, bit dos, bit tres,..., bit siete. Los cuatro primeros (del 0 al 3) se llaman LESS SIGNIFICANT BITS (LSB) o bits menos significativos, los otros (del 4 al 7) son los MOST SIGNIFICANT BITS (MSB) o bits más significativos . No os confundáis, no quiere decir que los LSB tengan menos significado o importancia que los MSB, es simplemente una forma de indicar que unos están colocados más a la izquierda, y por lo tanto tienen más valor aritmético, que los otros. Para nosotros son todos iguales, sólo que situados en distinta posición. A cada grupo de 4 bits, ya sean los LSB o MSB se les llama NIBBLE, o trozo en inglés, para indicar que se trata de un pedacito del byte. Cuando unimos dos bytes tenemos 16 bits numerados del 0 al 15, como podéis ver en la figura adjunta. A esta estructura se le denomina WORD, o palabra en inglés. Una palabra siempre indica 16 bits como se verá más adelante. El valor decimal de cada bit dentro del byte se basa en las reglas de la matemática binaria. En la tabla 1.1 podéis ver los valores decimales de sus correspondientes binarios en función de la posición del bit dentro del byte o de la palabra. Si unimos 2 palabras tendremos otra estructura más denominada DOUBLE WORD o doble palabra de 32 bits. Las más utilizadas, y las que veremos por ahora son el byte y la pala-

bra.
Como sabemos, los registros del microprocesador no son sino unas celdillas de memoria como las de la RAM pero situadas dentro del mismo para que a la hora de trabajar con ellos estén más accesibles. Por esta razón los registros presentan la misma estructura que hemos explicado anteriormente, es decir una unión de 2 bytes o palabra. Pero esta unión no es indivisible. Un registro de datos, como vimos, se puede llamar AX, BX, CX y DX, pero también se puede romper en dos registros de 8 bits, en aquellos casos en los que la operación que vamos a realizar no requiera utilizar un registro de 16 bits, con lo que ganamos registros. Los registros pasarían a llamarse igual pero con las letras H o L (High=alto, Low=bajo) indicando que parte del registro es la que estamos tratando. El registro AX se parte en sus dos nibbles correspondientes llamados AH y AL. De la misma forma tenemos BH, BL, CH, CL, DH y DL, con los otros 3 registros de datos.
Las instrucciones en código máquina contemplan esta posibilidad como vamos a ver en este mismo artículo, existiendo instrucciones para el manejo de registros de 8 y 16 bits, así como algunas que son exclusivamente para registros de 16 bits.

LA PRIMERA INSTRUCCION

Ya conocemos algunos de los registros que vamos a usar durante nuestro curso. Así que empecemos con alguna instrucción para realizar nuestro primer, pero pequeño, programa en lenguaje ensamblador.
Una de las operaciones que más se realizan dentro de nuestros programas en lenguaje ensamblador es la de mover datos de una lado para otro. Como sabemos, los datos e instrucciones de nuestros programas están en la memoria que es el banco de trabajo del microprocesador. Por lo tanto, tenemos que traer los datos desde ésta hacia la CPU y después volverlos a llevar, en los casos en los que sea necesario, de ésta hacia la memoria para que permanezcan allí hasta que hagan falta en un uso posterior.
La instrucción en lenguaje ensamblador que permite ésto se denomina MOV, abreviatura de MOVER (MOVE en inglés). La sintaxis es la siguiente:

MOV DESTINO, ORIGEN

La orden moverá el dato situado en ORIGEN hacia DESTINO, almacenándolo allí pero sin que por ello el dato situado en ORIGEN se borre o cambien. Es decir, tras esta operación tendremos el mismo número en ORIGEN y DESTINO.
Un ejemplo podría ser la orden:

MOV AX,BX

Si los contenidos de ambos registros son AX=1111h y BX=2222h antes de la operación, tras la ejecución de la misma por parte del microprocesador tendremos que AX=2222h y BX=2222h. Los registros de 16 bits como ya sabemos, terminan en X y pueden contener dos bytes. La notación utilizada es la hexadecimal que ya veremos en su momento. No os importe el contenido y la notación hexadecimal, simplemente y por ahora, mirad éste como un numero que nos sirve únicamente como ejemplo y nada más.
Volviendo al ejemplo, veis como el contenido de AX cambia porque se mueve hacia él, o se traspasa, el contenido del registro ORIGEN, en este caso BX, pero el registro BX sigue igual, no pierde su contenido. Esta orden de MOV, se entiende mejor si la llamamos COPIAR, porque en realidad es lo que se hace. Pero su nombre es MOVER y con él nos debemos conformar.
Las órdenes de ensamblador son bastante flexibles, es decir que en el caso de mover datos, que es nuestra primera operación, podemos hacerlo de todos los lugares a todos los lugares. Por ejemplo, tomando como base la orden anterior, la siguiente orden también está permitida:

MOV BX,AX

Esta, si os dais cuenta, es justamente la contraria de la anterior. Los registros AX y BX contendrían lo siguiente tras la operación: AX=1111h y BX=1111h, ¿los vais entendiendo?, seguro que sí.

Posición del bit dentro del byte	Valor decimal
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768

Tabla de equivalencias entre posiciones binarias y sus valores decimales

LOS ORIGENES Y LOS DESTINOS

Como hemos visto en el ejemplo anterior, los datos que hemos movido estaban en dos registros del microprocesador, o sea que en la operación no hemos movido nada entre la memoria y el microprocesador o viceversa. Para ello también contamos con un cierto número de posibles operaciones en la que el movimiento de datos de 8 o 16 bits puede llevarse a cabo con el banco de trabajo.
Con esto queremos decir que hay varios orígenes y varios destinos posibles de nuestros datos y son los que vamos a ver ahora.
Por un lado, como ya hemos visto, podemos mover datos entre dos registros de datos y por otro también podemos mover datos entre la memoria y uno de los registros o viceversa. Lo que no se puede hacer es mover datos entre dos posiciones de memoria sin

UNA DE LAS OPERACIONES QUE MAS SE REALIZAN EN LENGUAJE ENSAMBLADOR ES LA DE MOVER DATOS DE UNA PARTE A OTRA, PARA LO CUAL USAMOS LA ORDEN MOVE

que éstos pasen por uno de los registros previamente. Estudiemos esta afirmación con más cautela.

La orden MOV AX,BX está permitida, como también lo está la orden MOV AH, (1234h). La primera mueve un dato de registro origen BX al destino AX la segunda mueve un dato desde el destino, en este caso la dirección de memoria 1234h al registro AH. La cosa se complica, así que vayamos por partes.

Las posiciones de memoria (o direcciones de memoria) sabemos ya de partes anteriores de este curso, que son las etiquetas numéricas que ponemos en cada una de las "cajas de cerveza" con las que asemejabamos cada una de las partes de 8 bits en las que se divide la totalidad de la memoria de nuestro PC. Ahora, cuando queremos hacer uso de una de esas posiciones de memoria, simplemente le damos al ordenador el número que lo identifica, también llamado etiqueta de la posición de memoria. 1234h representa esa identificación a la que hacemos referencia, por lo que la instrucción anterior moverá el dato de 8 bits (byte) de esa posición de memoria a al registro AH, que, como hemos visto, es un registro de 8 bits. La posición de memoria se pone entre paréntesis para diferenciarla de lo que seria el dato 1234h. De esta forma las órdenes siguientes son diferentes y los contenidos de los registros tras ejecutarlas también lo son:

MOV AH, (1234h)

Mueve al registro AX el contenido de la posición de memoria situada en la posición 1234h. Si esta celdilla de memoria contiene el dato, por ejemplo, 56h el registro AH contendrá este número tras ejecutar la orden.

MOV AX, 1234h

Al no estar entre paréntesis, el ensamblador entenderá que lo que pretendemos es mover el número 1234h en el registro AX.

Los paréntesis indican CONTENIDO DE. En nuestro ejemplo CONTENIDO DE la posición de memoria 1234h. Si no los usamos el ensamblador lo traduce por un dato normal y corriente.

Hasta ahora hemos visto tres formas de mover los datos, veámoslas:

1) ENTRE REGISTROS

```
MOV AX,BX
MOV AH,BH
MOV DX,CX
MOV CH,DH
```

2) CON LA MEMORIA

```
MOV AH,(1234h)
MOV (1234h),AH
MOV AX,(1234h)
MOV DX,(1234h)
```

3) DIRECTO

```
MOV AX,1234h
MOV CX,2341h
MOV BX,8373h
```

8 ó 16 BITS

Una cosa que puede empezar a confundirnos y más vale volver a reiterar es la forma en que el microprocesador trabaja con los tamaños numéricos de BYTE y PALABRA (WORD). De la tabla de instrucciones que hemos visto arriba podemos sacar varias conclusiones.

La más importante de todas ellas sería la diferenciación que debemos hacer a la hora de trabajar con registros de 8 o 16 bits. Por ejemplo, el ensamblador generará un error si ponemos la siguiente orden en nuestro listado:

MOV AH,BX

Ya que intentamos mover el dato de BX, que es de 16 bits a un registro de 8 bits como es AH. El dato a mover es más grande que el lugar donde lo vamos a guardar luego no podemos.

Otra cuestión a tener en cuenta es que la orden...

MOV (1234h),(5678h)

no está permitida tampoco ya que, como dije, los datos han de pasar necesariamente por uno de los registros de datos. Para realizar la orden anterior, tendríamos que hacerlo en varios pasos:

```
MOV AX,(1234h)
MOV (5678h),AX
```

Usando el registro AX como almacén temporal del dato a traspasar.

En tercer lugar, de las posibles órdenes que podemos usar, sacamos otra conclusión más y es que si escribimos lo siguiente:

MOV 1234h,AX

también se produce un error, ya que intentamos mover el dato del registro AX a un número y no a una posición de memoria por no haber puesto los paréntesis que indican CONTENIDO DE.

Por último, no se si os habréis dado cuenta de un

lador para PC

LA ORDEN MOVE PERMITE MOVER DATOS HACIA LA MEMORIA Y DE ESTA A LOS REGISTROS DEL MICROPROCESADOR. TAMBIEN PERMITE INTRODUCIR UN VALOR CUALQUIERA DENTRO DE UN REGISTRO.

detalle. Cuando indicamos la orden MOV AX,(1234h), estamos diciendo lo siguiente al microprocesador:"Toma el contenido de la posición de memoria 1234 hexadecimal y lo metes dentro de el registro AX", pero ¿Si en una posición de memoria RAM sólo puede haber un byte (8 bits) como va ha rellenarse el registro AX que es de 16 bits?. La solución es que ante esta orden el microprocesador realiza un movimiento de 2 bytes en realidad. Concretamente los que están en las posiciones de memoria 1234h y 1235h, es decir la que indicamos en la orden y la que está justo a continuación de esta. Esto lo realiza de forma automática porque hacemos mención a un registro de 16 bits dentro del comando, en este caso AX. ¿Qué ocurriría si usamos la orden MOV AH,(1234h)?, pues que sólo un byte es movido de la memoria al registro AH, que es de 8 bits.

ALGUNAS ACLARACIONES

Hagamos un alto en el camino para descansar un poco y echar un vistazo atrás.

La orden MOV nos permite mover datos, cosa importantísima y muy necesaria dentro del ordenador. Los datos se pueden mover o copiar entre los registros y a/desde la memoria, pero siempre con el transcurso de los registros como intermediarios de la operación. Los datos tienen que ser de dos formatos; bytes o palabras. Los primeros son de 8 bits y caben exactamente en una posición de memoria RAM. Cuando necesitamos hacer operaciones con números más grandes de 255 tenemos que utilizar las palabras que nos permiten (ver tabla 1.1) usar números de 0 hasta 65535. El microprocesador siempre echará mano de los registros que están siendo usados en una orden en concreto para saber si los números con los que vamos a trabajar son de 8 o 16 bits. Por ello deberemos tener bastante cuidado con ordenes ambiguas del tipo MOV AX,(1234h) y MOV AH,(1234h).

EL PRIMER PROGRAMA

Bueno, ya llegó la hora. Vamos a hacer nuestro primer y simple programa en lenguaje ensamblador para PC. No será nada del otro mundo pero si que tendrá un significado especial para todos nosotros más adelante, cuando miremos atrás con cariño a nuestros primeros pinitos en el mundo de la programación.

Como seguramente habréis hecho, y si no es así todavía estáis a tiempo de corregir el error, habréis mirado la sección "herramientas: los debuggers" que salió en el número 1 y 2 de Código Juego, así como en este mismo número, porque vamos a realizar el primer uso de este pequeño depurador/ensamblador para nuestros trabajo con el assembly.

Iniciad DEBUG, introducid la orden A y pulsar ENTER. Ya estamos dentro del ensamblador que nos va a permitir trabajar con la primera orden que ya hemos estudiado; MOV. No os importe donde se va a introducir nuestro programa ensamblado, porque no vamos a guardar en disco nada de lo hecho, ya que serán sólo pruebas sin ningún valor salvo el educativo. Introduce las siguientes órdenes tal y como aquí aparecen.

(Nota: cuando se trabaja con DEBUG no hace falta añadir la h final que indica que el número está en hexadecimal ya que sólo utiliza ese formato, no permitiendo el sistema decimal o binario)

```
MOV AX,1111  -ENTER-
MOV BX,2222  -ENTER-
MOV AX,BX    -ENTER-
MOV (0300),AX -ENTER-
MOV CX,(0300) -ENTER-
-ENTER-
```

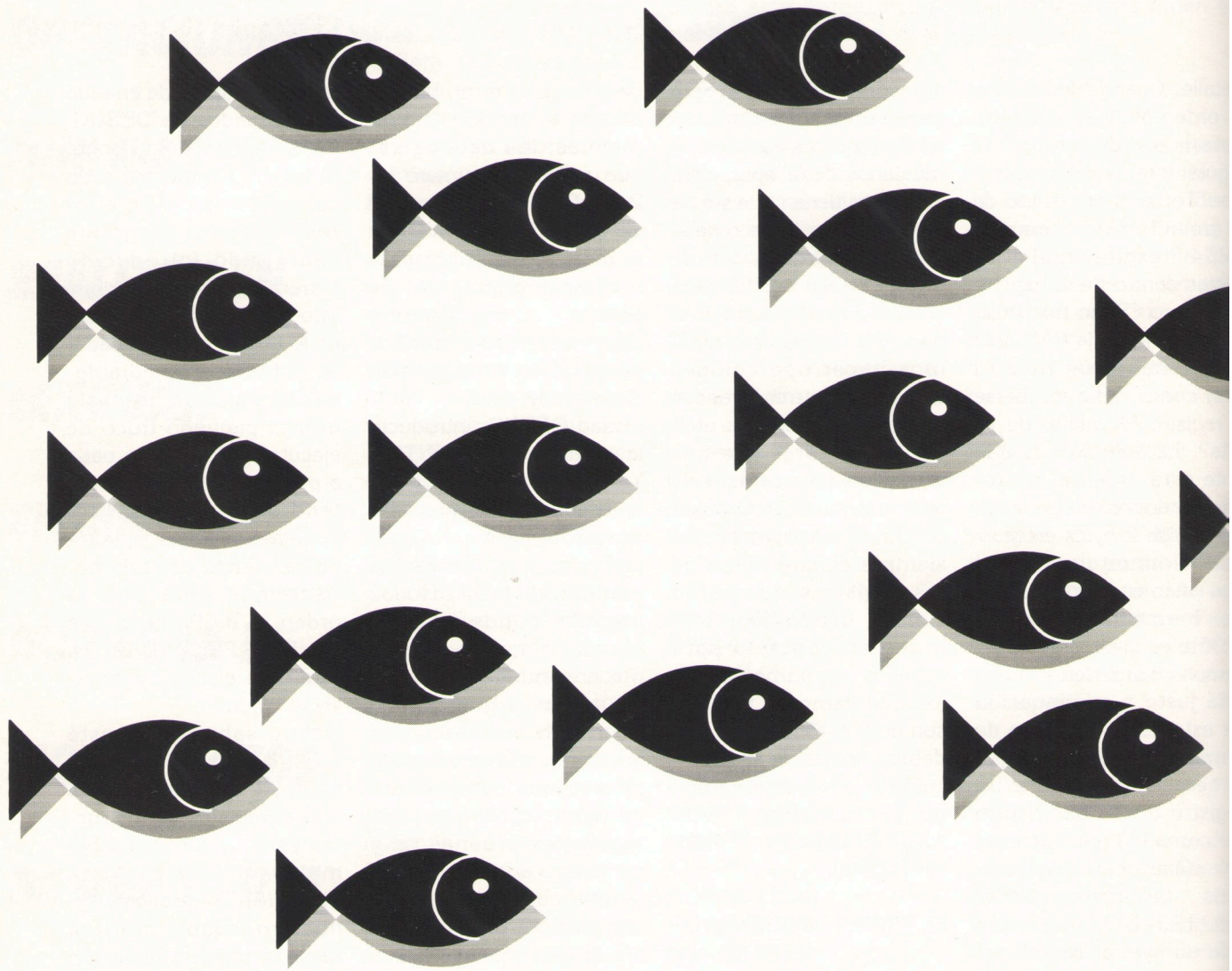
Al pulsar ENTER en la última posición del listado salimos del modo de ensamblado y pasamos a modo de espera de entrada de comandos de DEBUG. Bien, ya esta todo hecho. Si habéis introducido todo correctamente y sin errores, nuestro programa habrá sido introducido correctamente en memoria listo para ser corrido. Pero no lo hagáis o el ordenador se colgará irremediablemente. Para ello, vamos a usar el pequeño truco de ejecutar el programa paso a paso con lo que siempre tendremos el control y podremos estudiar el comportamiento del mismo. Usaremos para ello la orden P, de Proceed que significia PROCEDER, en este caso ejecutar y devolver el control.

Debug sabe donde está nuestro programa ensamblado, y apunta hacia él con el registro de instrucciones IP, que como sabemos indica la próxima dirección a ejecutar. En nuestro caso apuntará hacia la primera dirección de nuestro programa.

Por ahora ya hemos visto suficiente y es materia que necesita tiempo para entenderse. Espero que estudiéis un poco y prestéis un poco de atención. Al principio puede resultar un tanto lento y difícil pero pronto estaremos de lleno con programas más grandes y útiles.

Hasta el mes que viene.

No vaya en contra de todos...

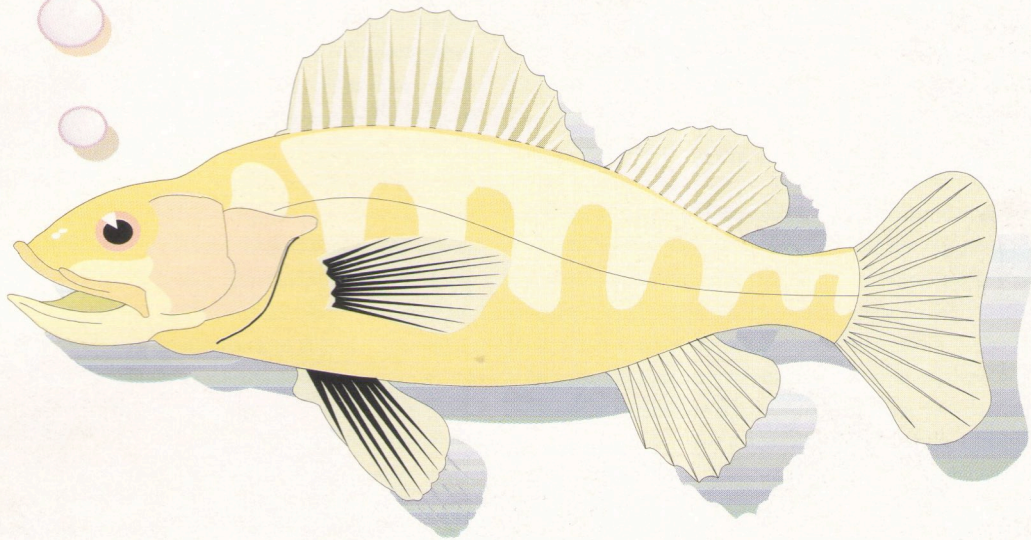


Asinfo

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

Asuntos de divorcio

Lector

Gonzalo Fuertes Fernandez

La informática me va a costar un buen disgusto. Me llamo Gonzalo y tengo 15 años. Desde hace aproximadamente cuatro tengo la fea costumbre de sentarme a los mandos de mi ordenador Amiga y llevarme largas y confortables horas sentado frente a él. Y digo lo de la fea costumbre porque poco a poco se está convirtiendo en eso. Me explico.

Los programas se amontonan y todos parecen gustarme, especialmente, como no, los juegos. Muchos de mis amigos, que no comulgan conmigo en cuanto a las aficiones informáticas, me tachan de atontado por preferir quedarme en casa con mi ordenador antes que irme con ellos a tomar unas copas a la disco de moda, pero esto nunca me importó en lo más mínimo. De hecho, era cierto que lo que más me llamaba la atención eran lo juegos que habían salido al mercado, como jugarlos y como acabarlos, así que porque narices tenía que ir a la calle a gastar dinero en tabaco y alcohol.

Esto fue perfectamente durante bastante tiempo, no apeteciéndoseme salir a la calle para nada y cada vez menos me atraía la discoteca donde mis amigos decían encontrar el paraiso de la diversión.

El sistema funcionaba genial para mi e incluso para mis padres, que ante mi actitud hogareña y recogida se alegraban de que no estuviese en la calle, fomentándome de vez en cuando con algún que otro juego o periférico que yo aceptaba gustoso. La familia estaba muy unida (yo soy hijo único) hasta que ella apareció en mi vida. Una esplendorosa rubia de ojos verdes que trabaja en una tienda de informática y a la cual conocí por la mano negra del destino que a veces nos juega buenas o malas pasadas.

Mi padre, siguiendo su politica de satisfacerme, informáticamente hablando, me había dado un suculento billete de cinco mil pesetas que unidas a las que yo había logrado ahorrar hasta ese momento completaban lo que necesitaba para adquirir un disco duro mayor. Tras hablar con Pepe el dueño de mi tienda habitual, éste me dijo que no tenía ese tipo de discos duros, así que me indicó otra tienda en el centro de Madrid. Cuando entré a preguntar, aquellos ojos verdes hicieron que mi ordenador pasase a un plano de importancia cero. Su forma de preguntarme que deseaba hizo que mi corazón galopase como un caballo indio. Jamás me he sentido tan ridículo en todos los días de mi vida, como cuando mi lengua empezó a trabucarse sin poder contestarle. De repente las palabras "disco duro" brotarón de mi garganta como por impulso, pero sin que en realidad supiese que había dicho exactamente. Ella contesto, "¿De qué tipo?", y ya ahí si que me costó lo mio, contestar sin medio tartamudear.

Aquella muchacha, a la que no compré el disco duro, puesto que como andaba justo de pelas, las necesitaba para ir a la disco donde ella iba, cosa que averigué gracias a Pepe, el de la tienda de informática habitual, resultó ser un encanto que poco a poco acabó enamorándome hasta los huesos.

Desde ese momento, iba a su tienda a comprar pequeñas tonterías que ni siquiera me hacian falta, cosa por la cual Pepe me hacía ver su enfado, ya que dejé de comprárselas a él. El ordenador empezó a coger polvo y mi desmesurada afición a la informática empezó a decaer.

Loli, que así se llama ella, fue progresivamente cayendo en mis redes hasta que finalmente, ella misma viendo mi indecisión me invito un día a su fiesta de cumpleaños. Pero, en realidad, lo que acabó de unirme a ella fué cuando me pregunto si yo podía enseñarle algo de BASIC e informática general, cosa que le ayudaría en gran medida a mejorar en su trabajo. Por supuesto, accedí. Tras dos meses, mi relación con ella se tambalea por lo mismo que nos unió, es decir los ordenadores. Ahora, que tengo una oferta de una compañía de software para hacer un programa para el Amiga y ello me toma mucho tiempo, ella se queja del poco tiempo que tenemos para nosotros. No me lo ha dicho todavía, pero el ultimatum..."el ordenador o yo" está a punto de salir de sus labios. Y me pregunto por qué es todo tan complicado y por qué la informática está reñida con las mujeres. El caso es que si vosotros, los de la revista, habéis tenido un problema como el mio os rogaría que me ayudaséis aunque sea moralmente, porque lo otro sé que tiene difícil solución.

Gracias de todas formas.

El informático enamorado.

La tintilla informática



El grito Hipohuracanado

¡Hola amigos lectores! Antes que nada, agradecer las numerosas cartas que hemos recibido para este apartado de Código Juego. Apelamos nuevamente a vuestra paciencia por no desesperar y seguir enviándonos vuestras quejas informáticas sin recibir respuesta por el momento, pero "todo llegará queridos amigos..."

¡Al grano Tintilla que te enrollas demasiado !

EXPOSICION DE LOS HECHOS

Nos escriben dos amigos de Donosti (Gipuzkoa) diciéndonos ,entre otras cosas, que son poseedores de un Amiga (concretamente el A500). Para los no iniciados diré someramente que el Amiga es un ordenador personal de la empresa Commodore, recordaréis quizás al C-64 de la misma marca que apareció en el mercado para hacer sombra al archiconocido ZX-Spectrum. Entre sus cualidades, que son muchas, destacaríamos el sonido y los gráficos.

Se quejan de que en nuestro curso de ensamblador sobre el microprocesador Motorola 68.000 , que si bien es el mismo para los ordenadores personales ATARI y AMIGA, lo referimos todo exclusivamente al ATARI, omitiendo al agraviado AMIGA.

Continúan lamentándose del precio de nuestra revista que según nuestros amigos Donostiarras es TOTALMENTE PROHIBITIVO para sus bolsillos, más aún cuando comprobaron que dentro no hacíamos alusión al A500, concluyeron que el precio fué definitivamente "un sablazo".

COMENTARIO

LLegados a este punto, quizás no entendáis aún la comparación entre esta queja con "Pepe Pótamo" de los inolvidables dibujos animados de la televisión. Pepe Pótamo era un hipopótamo de ficción que viajaba por el mundo en compañía de un monito salteando mil y una desventuras, rescatando a princesas desvalidas, ayudando a desgarbados aventureros en apuros... Al principio Pepe trataba de razonar con los malos para que liberasen a tal o cual persona, pero siempre acababa sufriendo las consecuencias de los "pillos" que solían cambiar a la dama por Pepe y su amigo. Para salir con éxito de la olla de los caníbales o de las mil y una calamidades por defender una causa noble, usaba su GRITO HIPOHURACANADO que consistía en tomar aire una y otra vez hasta que su cuerpo engordaba a casi el triple y expulsarlo por la boca en forma de huracán dejando a los malandrines a buen recaudo.

Al leer cuidadosamente estas cartas, tuvimos que implorar un "mea culpa" sin dar opción a un nuevo grito de Pepe Pótamo que nos dejara maltrechos. Es totalmente cierto que si el Motorola 68.000 lo comparten el ATARI y EL AMIGA, existen enormes diferencias entre ambos, a modo de ejemplo nombrar que el tratamiento de los periféricos son diametralmente diferentes, las interrupciones ídem de ídem... Por ello me alegré enormemente cuando al analizar las quejas de estos incondicionales (espero) llegamos a la conclusión de incluir un curso de ENSAMBLADOR ESPECIFICAMENTE PARA AMIGA, dando respuesta a nuestros lectores.

El precio ¡Uf! Cuando nos planteamos seriamente salir al mercado nacional con nuestra publicación, pensamos en un único objetivo : Un producto de calidad que satisfaciera a los miles de usuarios informáticos hartos de la informática de gestión y del comentario de video juegos sin un fin práctico.

Si os fijáis miréis donde miréis estamos invadidos por las casas publicitarias. Nuestra revista lleva poca publicidad a cosa hecha, analizar por gusto el resto de publicaciones, os llevareis un desagradable sorpresa. Somos tremendamente selectivos a la hora de introducirla, pues de nada nos valdría tener 60 paginas de ella y solamente 8 de contenido a un precio formidable. Ofrecemos casi 68 paginas repletas de contenido práctico que no variaremos por nada ni por nadie, es más nuestra intención es no saturaros jamás de anuncios publicitarios, y eso obviamente debe redundar en el precio final.

Los cursos pueden resultaros más o menos interesantes, algunos tendréis los conocimientos informáticos como para no necesitarnos hasta el número 15, pero siempre nos podréis usar de recordatorio o repaso, otros diréis que casi no llegáis a lo que tratamos de enseñamos, en definitiva nunca llueve a gusto de todos.

Finalmente cuando te adentras en la informática y miras atrás, observas que tu biblioteca tiene un 60% de libros en inglés, porque si adquieres la versión castellana los errores de traducción se suceden a la velocidad del rayo. Podríamos extraer a modo de ejemplo de una versión pirata del PCTOOLS que alguien ha traducido y que al finalizar nos dice "Felipe pasó del Inglés". Los libros sean de la índole que sean no son especialmente baratos (por desgracia) y pienso que nuestro precio quizás sea algo elevado es posible, pero si pensáis:

- El equipo humano que ponemos a vuestra disposición.
- El tiempo empleado en el desarrollo de cada uno de los temas y cursos que ofrecemos.
- Los medios técnicos necesarios para llegar hasta vosotros.
- Somos autodidactas, no pertenecemos a ninguna Multinacional o macroempresa que nos ampare.
- La escasa publicidad como lema.

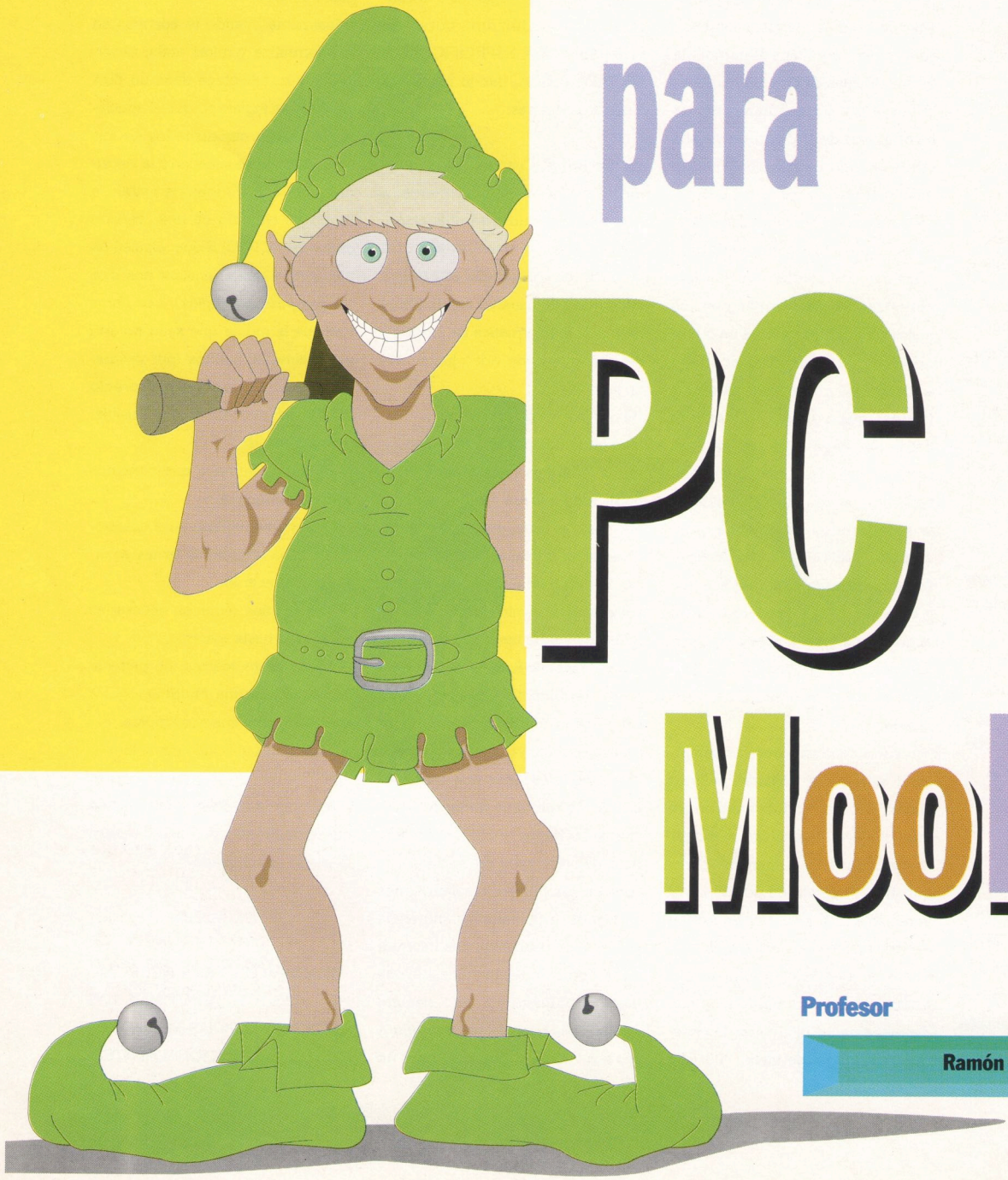
Llegaríais a la misma conclusión que la redacción de código Juego. El precio es adecuado.

MORALEJA

MEJORAMOS PARA ACERCARNOS MAS A TI. ADMITIMOS LAS CRITICAS Y GRACIAS POR ADQUIRIRNOS INCONDICIONALMENTE.

CURSO PRACTICO

Así se hace un juego para PC Mookie



Profesor

Ramón López

Ya estuvimos indagando en el número anterior en que consistía el fundido de colores. En realidad se trata de un truco visual que se consigue disminuyendo la intensidad de la luminosidad del color original en pantalla o aumentándolo dependiendo de si se trataba de un fundido a negro o a color. Como vimos, los colores aparecen en pantalla gracias a la iluminación con mayor o menor fuerza de los tres componentes de color en los que se divide cualquiera de los que forman un gráfico en la pantalla, en nuestro caso el logotipo de MOOKIE, y que se conocen como RGB o Rojo, Verde y Azul. La intensidad de dichos tres componentes que dan vida a cada color se mide con números que oscilan entre 0 y 255.

Vimos, también, un ejemplo con un color escogido al azar. En nuestro caso fue el color blanco, el cual tiene la máxima cantidad de los tres colores, es decir, 255 en los tres casos. El color bajaba de intensidad a medida que hacíamos disminuíamos la cantidad de componente verde del mismo, convirtiéndose en un color rosaceo. Este ejemplo nos sirvió de maravilla para ilustrar como llega a conseguirse este efecto. Pero pasemos a estudiarlo en el propio programa MAIN.ASM que venia en CJ1.

EL LISTADO

Antes que estudiar como hemos logrado plasmar en código máquina el fundido de color de las diferentes pantallas de presentación de juego, tenemos que hacer un alto en el camino para estudiar como se carga un archivo desde disco hacia la memoria en el lugar que nosotros deseemos, para utilizar después los datos cargados a nuestro antojo, en este caso los que corresponden a la información gráfica de las pantallas y la paleta de colores de cada una de ellas. Incluso me veo obligado a estas alturas a dar un paso atrás y explicaros como se han conseguido las pantallas que habéis visto en el número 1 de Código Juego.

Para crear una pantalla gráfica existen multitud de aplicaciones en mercado. Por nombrar algunas diría Paintbrush, Photostyler, 3D Studio, Corel Draw, etc. Con estas aplicaciones de logran pantallas de gran calidad gráfica que podemos utilizar en nuestros juegos. En general los gráficos se pueden crear de tres formas; manual, renderizados y escaneados. La manual es la más complicada pero la que nos permite una mayor libertad en cuanto a calidad e imaginación. La renderización consigue gráficos electrónicos de gran calidad tridimensional pero siempre tienen ese olor a sintético. Por último el escaneo de imágenes nos da la realidad total. Estos procesos se pueden combinar para darnos gráficos de todos los gustos.

Pero, no es tema de esta sección la creación gráfica de las imágenes. Partimos, pues, de que ya tenéis las pantallas creadas o de que tenéis un amigote que las ha creado.

Las imágenes en el PC se guardan en disco en muchos formatos diferentes como el PCX, IMG, TIF, GIF, TGA, JPEG, etc, pero todos ellos reunen los 3 bloques de información fundamentales para cada una de ellas. Estos son:

1) Datos gráficos: Es decir la información de los pixels que forman el gráfico
2) Paleta: Los componentes de color RGB de cada uno de los colores (256 en el modo gráfico que nos interesa)
3) La información de formato: Tamaño de la imagen en filas y columnas de pixels, número de colores (256, 32768, 65535, 16 millones...)

En el modo 13h, 256 colores 320x200 pixels tendremos una paleta que medirá 768 bytes, es decir tres bytes RGB por color por 256 colores diferentes. Además, si tenemos 320x200 pixels nos da un total de 64000 bytes de información gráfica. El problema que se nos presenta es el de que formato elegir para guardar la información de los gráficos en disco.

Para ello hemos elegido lo que se conoce como formato crudo o sin compresión. Ello se debe a que si utilizamos un formato con compresión de datos, como alguno de los que he mencionado arriba, tendremos que descomprimirlos en nuestro programa, cosa que estudiaremos más adelante. Esto tiene la gran ventaja de ahorrar espacio en disco, pero nos complicaría exageradamente la existencia en estos niveles de comienzo en los que nos hayamos.

El formato .PAN, o pantalla, en el que están guardadas la pantallas de presentación de MOOKIE que podéis encontrar en el disco de CJ1, es el formato crudo o sin compresión del que os he hablado y que se resume de la siguiente manera:

1) 768 bytes de la paleta de colores
2) 64000 bytes de información de pixels

Tamaño del archivo 64768 bytes.

Como podéis ver existe un archivo para cada pantalla de presentación, con nombres como MOOKIEPR.PAN, MARCOS.PAN y RAMON.PAN. Si los miráis internamente con cualquier programa que permita ver la información hexadecimal como LIST.COM que explicamos en CJ2, veréis que los primeros 768 bytes del archivo presentan una repetitividad de tres en tres correspondientes a los bytes R, G y B de la paleta de colores.

Una vez que entedéis esto, vamos a pasar a como se carga este archivo en memoria para que la información gráfica pueda ser utilizada en el juego MOOKIE.

LA CARGA DE ARCHIVOS

Cuando un archivo se ha grabado en disco y se pretende recuperar los datos que hay en él, tenemos que echar mano al MSDOS y sus funciones handle de manejo de archivos.

Es pretension de este apartado obviar lo innecesario, por lo que no se explicarán aquí las obsoletas funciones FCB que ya no se utilizan, gracias a la manejabilidad de las más modernas y utilizables funciones handle.

Vayámonos a la parte del listado que está etiquetada como carga y que podéis ver en la imágen adjunta. Esta subrutina se encarga de cargar los archivos en memoria en función de una serie de varibles que están encadenadas en la parte del segmento de datos bajo el nombre de BLOQUES. ver figura.

La variable BLOQUES contiene toda la información que se necesita para cargar un archivo en memoria y que se explican, también, en el pequeño listado de MOOKIE.INC en la página 21 de CJ1. Estos datos son:

1) Nombre de fichero: 8 caracteres del nombre y 3 de la extensión.
2) Byte ASCIIZ que indica fin del nombre del archivo. La Z al final de ASCII significa que el dato es siempre un cero.
3) Puntero de carga (Word alto y bajo): indican un desplazamiento dentro de archivo desde donde empezaremos a cargar datos en caso de que no queramos que empieze desde la posición inicial del archivo.
4) Lugar en memoria: (Word bajo, byte alto) Sitio en la memoria donde queremos colocar los datos cargados, en formato Segmento:Offset.
5) BCOMPR: Tamaño del bloque a cargar en caso de que no este comprimido. Por ahora y como hemos indicado, el bloque no estará comprimido. Por tanto aquí siempre indicaremos un dato.
6) BDESC: Tamaño del bloque comprimido. No se usa todavía.
7) FLAG: Un byte que indica si el archivo pedirá que se le asigne memoria y por tanto indica que puede ser cargado en cualquier lugar de la misma, con lo que se obvia la variable "lugar en memoria", la cual se usa cuando queremos que los datos cargados sean introducidos en lugar prestablecido de la memoria.

Como véis, CARGA permite trabajar con cualquier tipo de archivos, comprimidos o no, y colocar un número X de datos en cuaquier posición de la memoria.

Pero estudiemos más a fondo las partes integrantes de la variable BLOQUES.

El nombre del fichero sigue las convenciones normales de MSDOS, es decir los ocho caracteres que se usan para el nombre en sí, el punto y después los 3 caracteres de la extensión. Se ponen entrecomillados para que el ensamblador



En las pantallas se puede apreciar como el color se va degradando paso a paso para conseguir el efecto de fundido. Las pantallas ocupan 64768 bytes en formato descomprimido hasta que aprendamos a comprimirlos con rutinas especiales de compresión gráfica.

sepa que estamos tratando con información alfanumerica y que de traducirlos a sus correspondientes códigos hexadecimales a la hora de compilar.
El byte 00h, que sigue al nombre es llamado byte ASCIIZ en la documentación de Microsoft. No sirve para nada, salvo para indicar a las funciones handle del sistema operativo que ahí se encuentra el final del nombre del archivo, por tanto no debemos saltárnoslo o nuestro programa dará errores.
El puntero de carga, siguiente variable, es de una utilidad innegable, ya que nos permite cargar datos desde un archivo pero desde la posición dentro del mismo que deseemos. Así, cuando queramos cargar los datos de la información gráfica de nuestras pantallas, que se encuentran justo después de los 768 bytes de la paleta de colores, le indicaremos a CARGA que desplace el puntero de carga ese número de bytes dentro de cada archivo de pantalla como veremos más adelante.
El lugar en memoria indica en que parte de la memoria convencional o base del PC vamos a introducir los datos a medida que se vayan cargando. Esto nos permite situar la información en el lugar que queramos para su utilización posterior. En el segmento de datos de MAIN.ASM podéis ver que hemos destinado un lugar especial para los datos de la paleta y que hemos etiquetado con el nombre de DACTABLE1. Cuando cargemos los datos de la paleta, indicaremos a CARGA que queremos cargar esos 768 bytes en esta posición de la memoria y no en otra.
En BCOMPR indicamos el tamaño en bytes que vamos a cargar en memoria y por último en FLAG indicamos que el bloque de datos deberá ser introducido en cualquier lugar de la memoria, en realidad el primero que esté disponible, tras pedirle al MSDOS que le asigne memoria como veremos ahora.

SUBRUTINA CARGA

Para llamar a CARGA deberemos cargar en el registro SI con la dirección de la memoria donde se encuentre la variable BLOQUES en cuestión, la cual como hemos visto anteriormente, reune toda la información sobre el archivo a cargar. Lo hacemos de la siguiente forma:

```
MOV SI, OFFSET BLOQUE3
CALL CARGA
```

De esta forma le pasamos a la subrutina CARGA todos los datos del archivo a cargar, en este caso BLOQUE1. Pero analicemos más detenidamente que contiene esta varible en concreto para que nos sirva de ejemplo.
BLOQUE3, para empezar, en el campo "nombre de archivo" tiene el nombre de la pantalla que queremos cargar en memoria, en este caso "MOOKIEPR.PAN". Como dijimos antes 8 caracteres, punto, 3 caracteres y para finalizar el byte ASCIIZ que como véis siempre vale 0. A continuación, nos dice que el lugar donde quiere comenzar a tomar datos desde el comienzo del archivo MOOKIEPR.PAN es el 0,0 luego justo desde el principio del archivo. Si lo comprobáis, cuando queremos cargar la información de los pixels de la pantalla en este campo aparecen los datos 768,0 como en la variable BLOQUE8 por ejemplo. Los datos siempre se guardan de forma byte bajo, byte alto luego ese dato será 768. De esta forma indicamos que queremos empezar a cargar desde el dato colocado en el lugar 768 dentro del archivo en cuestión.
A continuación, dentro de BLOQUE3, indicamos el lugar en la memoria donde queremos guardar los datos cargados. En esta ocasión están marcados como:
OFFSET DACTABLE2,SEG DATA. De nuevo posición baja, posición alta, recordad eso siempre. Los datos serán cargados en el segmento de datos justo en el lugar etiquetado como DACTABLE2.
El número de datos a cargar según indica BLOQUE3 es de 768 bytes según indican los datos 0,768 que vienen a continuación. Después sólo queda el FLAG que nos dice que no quiere pedir memoria para colocar los datos puesto que ya tiene sitio donde colocarlos, o sea DACTABLE2 que ya fue indicado en el campo anterior.
Pues bien, esta es la forma interna de la variable BLOQUES que está en el segmento de datos de nuestro programa. Veamos ahora como se cargan los datos en la memoria con la subrutina CARGA. Al principio de la subrutina, pasamos el valor de entrada que venia en SI, es decir el comienzo en memoria de la varible que nos indica que tenemos que cargar, al registro DX y a continuación metemos en el registro AX el valor 03D00h, para llamar a las funciones del MSDOS.
La función a la que llamamos es la 3Dh que se encarga de abrir un archivo y y no ocurre ningún error asignarle un handle que es como una tarjeta identificativa y que utilizaremos a partir de ahora para referirnos a él. Si ocurre un error en la apertura del archivo,

generalmente porque el disco esté dañado o que en realidad el archivo no se encuentre, de esta interrupción volvemos con el acarreo alto por ello hacemos la comprobración JNC LOAD_1, la cual, si falla, nos conduce a una salida del programa hacia la subrutina EXIT que se encargará de devolvernos al MSDOS con un mensaje de error de "archivo no encontrado". Si como debe ser la comprobación es correcta nuestro programa continua en la etiqueta LOAD_1.

El handle o llave identificativa del fichero es devuelto en el registro AX y para no perderlo en las sucesivas operaciones, se lo pasamos a BX.

La siguiente función del MSDOS que vamos a utilizar por medio de INT 21h es la 42h. Con ella logramos situar el puntero de carga en la posición del archivo desde la que queremos empezar a tomar datos. Para ello debemos tener esa posición cargada en los registros CX y DX antes de llamar a esta interrupción. Para ello los tomamos de las posiciones 13 y 15 de la variable bloques. Echale un vistazo por si tienes dudas a la explicacion anterior y comprueba las posiciones de las que te hablamos. Si esta función da error no tiene más remedio que ser que el fichero esté dañado o corrupto por lo que saltamos a dicho error.

A continuación, y si el FLAG de petición de memoria está encendido, llamamos a la subrutina ASIGNA la cual buscará cual es la siguiente posición libre en la memoria del PC para dárnosla a nosotros e indicándonos cual es el comienzo de la misma. Si hay un error, porque no hubiese memoria suficiente, saldríamos al MSDOS con el consiguiente mensaje que nos indica que debemos liberar alguna. De todas formas esto es difícil que ocurra ahora que nuestro programa es pequeño pero es algo que deberemos tener siempre en mente.

Tras esto comprobamos si está comprimido el fichero y en caso contrario, y por ahora va a ser así, saltamos a la etiqueta LOAD_6 donde cargamos el archivo y tras su utilización procedemos a cerrarlo liberando el handle correspondiente con la función 3Eh de la interrupción 21h.

El archivo, o la parte de él que queremos, ha sido cargado en memoria listo para su utilización. Como habéis comprobado esta rutina puede ser, por su estructura, utilizada para cargar no sólo las pantallas, sino todo lo que queramos dentro de nuestro programa. Te recomiendo que la estudies detenidamente y que te apoyes en tu estudio con algún que otro libro donde te expliquen los servicios del MSDOS. También te recomiendo que, si no lo has hecho hasta ahora, empieces a estudiar la sección "El MSDOS y el programador de Videojuegos" donde encontrarás toda la información, paso a paso, de como se puede acceder a estas funciones de las que vamos a ver muchas de ellas aquí, de forma práctica. De hecho, ya hoy, has aprendido algunas de ellas.

En el próximo Cj nos veremos con más cosas.

```
;*****************************************************************************
;Modulo: CARGA
;descripcion: Carga un archivo o un bloque de dentro de un archivo en la memoria
;             El archivo o bloque puede estar comprimido o no.
;Entradas: SI debe apuntar a la variable BLOQUES que indica que hay que cargar
;          donde hay que cargarlo y si esta comprimido o no.
;       Ramon Lopez, 21 Nov 1992
;*****************************************************************************

                Name CARGA
            INCLUDE MOOKIE.INC

CODE        SEGMENT         PUBLIC          'CODE'
            ASSUME  CS:CODE,DS:DATA,ES:DATA

CARGA       PROC    NEAR
            MOV     DX,SI
            MOV     AX,03D00H                       ;APERTURA DE ARCHIVO PARA LECTURA
            INT     21H
            JNC     LOAD_1
            MOV     ERROR,4                         ;ERROR DE APERTURA DE ARCHIVO
            JMP     EXIT

LOAD_1:     MOV     BX,AX                           ;GUARDA EL HANDLE
            MOV     CX,[SI+15]
            MOV     DX,[SI+13]                  ;SITUA EL PUNTERO DENTRO DEL FICHERO
            MOV     AX,04200H                       ;DESDE EL COMIENZO DEL MISMO
            INT     21H

            JNC     LOAD_2
            MOV     ERROR,5                         ;ERROR DE CORRUPCION DEL FICHERO
            CLC
            JMP     EXIT

LOAD_2:     MOV     AL,[SI+25]                      ;FLAG DE PETICION DE MEMORIA
            AND     AL,AL
            JZ      LOAD_3
            PUSH    BX                              ;GUARDA EL HANDLE
            MOV     BX,[SI+23]                      ;BYTES TOTALES QUE NECESITA EL BLOQUE EN MEMORIA
            CALL    ASIGNA                          ;LLAMA AL ASIGNADOR DE MEMORIA
            POP     BX                              ;RECUPERA EL HANDLE DEL FICHERO
            JMP     SHORT LOAD_4

LOAD_3:     LES     DI,[SI+17]                      ;LA POSICION DE CARGA NO ESTA YA ASIGNADA
LOAD_4:     MOV     CX,[SI+21]
            AND     CX,CX                           ;SE TRATA DE UN FICHERO COMPRIMIDO?
            JZ      LOAD_6
            MOV     AX,0A000H                       ; CARGA EN MEMORIA DE VIDEO
            MOV     DS,AX
            XOR     DX,DX
            MOV     AH,3FH
            INT     21H
            JNC     LOAD_5
            MOV     ERROR,5                         ;ERROR DE CORRUPCION DEL FICHERO
            CLC
            JMP     EXIT

LOAD_5:     CALL    DESCOMP
            JMP     SHORT CIERRA

LOAD_6:     MOV     CX,[SI+23]                      ;SE CARGA EL ARCHIVO DESCOMPRIMIDO
            PUSH    DS
            PUSH    ES
            POP     DS
            MOV     DX,DI
            MOV     AH,3FH
            INT     21H
            POP     DS
            JNC     CIERRA
            MOV     ERROR,5
            CLC
            JMP     EXIT


CIERRA:     MOV     AH,3EH
            INT     21H                             ;CIERRA EL ARCHIVO, EL HANDLE YA ESTA EN BX
            RET
CARGA       ENDP

ASIGNA      PROC    NEAR
            SHR     BX,1
            SHR     BX,1
            SHR     BX,1
            SHR     BX,1
            INC     BX
            MOV     AH,48H                          ;PETICION DE MEMORIA
            INT     21H
            JNC     ASIGNA_1
            MOV     ERROR,3                         ;ERROR DE FALTA DE MEMORIA
            CLC
            JMP     EXIT

ASIGNA_1:   MOV     ES,AX
            MOV     SG_CARGA,AX
            XOR     DI,DI
            RET
ASIGNA      ENDP
DESCOMP     PROC
DESCOMP     ENDP
```

Gráficos 3D

Colaborador

Alberto García

EL SECRETO DE LAS 3D

El monitor del ordenador es un plano en 2 dimensiones correspondientes al alto y al ancho. Para identificar un punto en ese plano, usamos pares de valores que representan la altura (a la que a partir de ahora llamaremos Y) y la posición horizontal (a partir de ahora llamaremos X) que ocupa dentro del plano.

El primer punto del monitor es el que tiene valores x=1 e Y=1. Para mayor comodidad lo representamos como (1,1). La forma general para hacer referencia a un punto del plano, es (X,Y).

Este sistema para nombrar puntos, se llama CARTESIANO el cual se puede aumentar aún en una dimensión más, la profundidad, que a partir de ahora llamaremos Z. Pero, ¿cómo podemos representar un punto (X,Y,Z) en el monitor si este es un plano de 2 dimensiones?, fácil, usando la cabeza y un poco de matemáticas.

LAS MATEMATICAS Y LAS 3D

No te asustes, las matemáticas necesarias son muy básicas, a nivel de BUP. Y si aún así no las comprendes, no te preocupes, ya que vamos a deducir unas fórmulas y sólo tendrás que ponerlas en tus programas aunque no las comprendas.

Si miras atentamente el dibujo1, te darás cuenta de que entre el objeto, el horizonte y tus ojos, se forma un triángulo, que a su vez es proporcional al triángulo que se forma entre tus ojos, el punto de corte con el plano de la pantalla de la línea imaginaria que une al objeto con tus ojos y al punto de corte con el plano de la línea que unen tus ojos con el horizonte.

Por lo tanto, según el principio básico de la semejanza de triángulos de la trigonometría:

Y/Z=SY/D

y despejando:

SY=(Y*D)/Z

Y como SY representa la intersección con la pantalla de la linea visual, SY es la posición Y de la pantalla, fácil ¿no? Aclaremos esto antes de pasar a deducir la fórmula para conseguir la posición X de la pantalla, que D es lo que los fotógrafos, astrónomos, y otras personas que trabajan con aparatos ópticos llaman distancia focal.

Por lo general un valor de 1200 es un valor aceptable, ya que si no el objeto puede salir algo deformado.

La coordenada X en pantalla se deduce exactamente igual que la Y, la única diferencia es que hay que imaginar que vemos el dibujo1 desde arriba, o sea, al observador sólo se le vería la coronilla, el objeto observado tendría un desplazamiento respecto a la linea de visión (mirar dibujo2). Las formulas, al igual que anteriormente son: X/Z=SX/D de donde: SX=(X*D)/Z por lo tanto, la posicion (X,Y) en la pantalla de el punto (X,Y,Z) es (SX,SY) donde SX=(X*D)/Z y SY=(Y*D)/Z. Sin embargo, esto no nos basta, sigue leyendo.

LAS FIGURAS EN 3D

Ya sabemos como dibujar un punto (X,Y,Z) en la pantalla, pero ¿cómo dibujamos una linea? Bien, sabemos que una línea es una sucesión de puntos, por lo tanto, lo lógico sería calcular muchos de esos puntos y representarlos. En teoría es correcto, pero en la práctica, no es factible debido a que al sumar el tiempo necesario para calcular cada punto el proceso se haría lentísimo, por lo tanto, tendremos que buscar un atajo. Sabemos que entre dos puntos hay una y sólo una recta posible que pase por ambos puntos, por lo tanto, nos basta con dos puntos para dibujar la recta. El proceso es el siguiente, se calcula la posición en pantalla de los extremos del segmento y despues se unen con una línea normal en 2D (usando la instrucción normal que tenga el lenguaje que uses para dibujar líneas). Por ejemplo si nuestro lenguaje usa la instrucción LINE para dibujar una línea y queremos dibujar la línea que va desde (X1,Y1,Z1) a (X2,Y2,Z2) podríamos hacer:

```
SX1=(X1*D)/Z1 SY1=(Y1*D)/Z1
SX2=(X2*D)/Z2 SY2=(Y2*D)/Z2
LINE (SX1,SY1,SX2,SY2)
```

Para dibujar figuras más complejas, sólo tendremos que usar líneas más unidas.

LA REPRESENTACION EN PANTALLA

Los objetos en el espacio, ocupan lugares diferentes, es decir podemos decir que el vaso está encima de la mesa o que la caja de zapatos está boca abajo. Cuando representamos un objeto en la pantalla tambien hemos de indicar la posición de este y tambien si está boca abajo o en cualquier otra posición.

El proceso de indicar el lugar en el cual se halla el objeto, se llama TRASLACION. Consiste simplemente en indicar en que posición del sistema de referencia (ejes X,Y,Z) se halla. Por ejemplo, podríamos decir que el vaso que está sobre la mesa, está 2 metros delante de nosotros (Z) y a la altura de 1 metro (Y). Las operaciones para trasladar un objeto a cualquier punto del espacio, son muy sencillas, consisten en sumar a cada componente X, Y, y Z el valor del desplazamiento. En el ejemplo del vaso, tendríamos que:

```
X=X Y=Y+1 Z=Z+2
```

Generalizando queda:

```
X=X+DX Y=Y+DY Z=Z+DZ
```

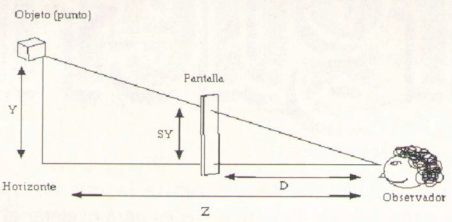
Pero aún queda el proceso más complicado de todos. Se trata de indicar si el objeto esta boca abajo, boca arriba o sobre un lateral. Este proceso se llama GIRO. Para mayor comodidad lo descompondremos en tres giros simples, uno por cada eje. Para no complicar demasiado, pasaré a daros las matrices que definen cada uno de los tres giros y despues deduciremos las fórmulas, pero antes hemos de explicar como podemos medir los ángulos de giro. Como todos sabemos la circunferencia está dividida en 360 grados, ésta es una medida arbitraria, sin embargo es la más usada en Europa (y en España). Sin embargo no es la más adecuada para realizar cálculos por ordenador. Es más conveniente la medida en radianes. Explicar la medida en radianes sería muy largo, así que daremos la razón de conversión de grados a radianes. De todas formas es conveniente saber que media circunferencia (180 grados) corresponde al numero PI (3.1416) y por lo tanto toda la circunferencia (360 grados) corresponden a 2*PI. La razon de conversión es:

```
RAD=(GRA*PI)/180
```

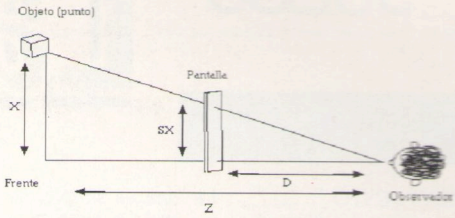
Ahora ya estamos preparados para deducir las fórmulas de las matrices, Las matrices son las siguientes:

GIRO EN EJE Y:

```
( COS SEN 0 0 ) (X,Y,Z,1) ( -
```



Dibujo 2. Determinacion de Y



Dibujo 1. Determinacion de X

SEN COS 0 0) (0 0 1 0) (0 0 0 1)
GIRO EN EJE Z:
(COS 0 -SEN 0) (X,Y,Z,1) (0 1 0 0) (SEN 0 COS 0) (0 0 0 1)
GIRO EN EJE X:
(1 0 0 0) (X,Y,Z,1) (0 COS SEN 0) (0 -SEN COS 0) (0 0 0 1)
Siendo Ax el ángulo de giro en el eje X, Ay el de el eje Y y Az en del eje Z, tenemos algebraicamente las siguientes fórmulas deducidas de las matrices:

XA=COS(Ay)*X-SEN(Ay)
ZA=SEN(Ay)*X+COS(Ay)*Z
X=COS(Az)*XA+SEN(Az)*Y
YA=COS(Az)*Y-SEN(Az)*XA
Z=COS(Ax)*ZA-SEN(Ax)*YA
Y=SEN(Ax)*ZA+COS(Ax)*YA
Donde X,Y,Z son los resultados finales después de haber realizado el giro, y XA,ZA,YA son variables para guardar cálculos temporales.

UNIENDO TODAS LAS PIEZAS

Ahora sabemos como proyectar un punto de 3D a 2D, sabemos hacer una traslación y un giro, pero ¿cómo se pone en pantalla un gráfico en 3D? El proceso es simple, primero tomamos los datos de tres en tres que representan a cada punto que definen un objeto, por supuesto los objetos se definen alrededor del centro de coordendas (luego mediante la traslación se colocarán en el lugar que queramos). Seguidamente, estos han de ser girados de forma que si tenemos (X,Y,Z) y lo hacemos girar, tendremos como resultado otros tres puntos que son los mismos pero ya girados:

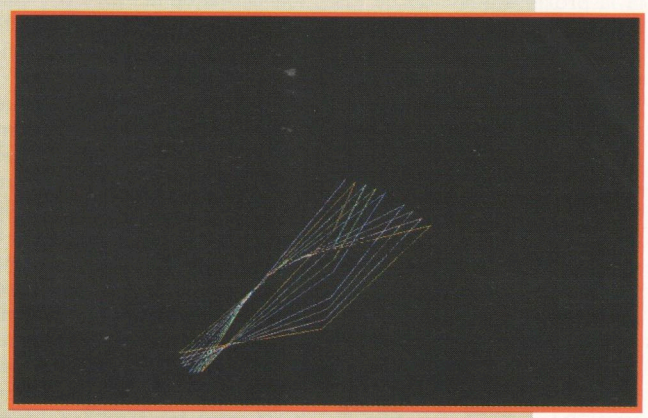
(X,Y,Z) —GIRO—> (X,Y,Z)
El siguiente paso es hacer la traslación, de forma que obtenemos la misma terna de puntos ya trasladados:

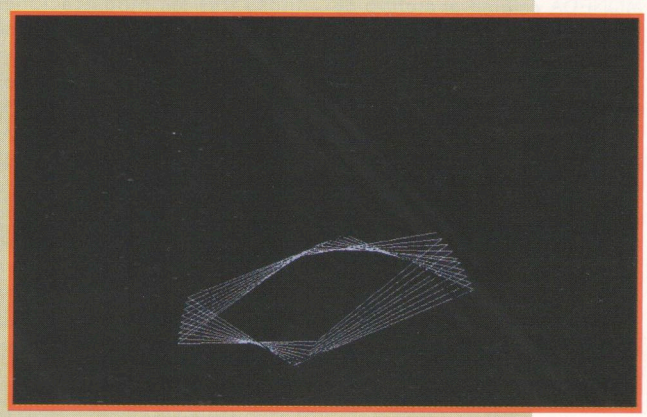
(X,Y,Z)—TRASLACION—> (X,Y,Z)

Y finalmente aplicamos la proyeccion a 2D:

(X,Y,Z) —PROYECCION—> (SX,SY)

obteniendo así un par de





PARA FINALIZAR

Listado DEMO.PAS en pag.66

La virguería

Colaborador

Francisco Vinuesa

MOVIMIENTO

Sin lugar a dudas, el movimiento de objetos por la pantalla es una parte fundamental de la mayoría de los tipos de juegos. Es cierto que en juegos de mesa como el ajedrez o algunos juegos de ROL esto puede ser trivial pero en los de tipo PLATAFORMA o ARCADE se torna algo vitál.

Igual que existen tipos de juegos diferentes, vemos que hay muchas maneras de mover estos objetos en pantalla. Si observamos el movimiento de las naves enemigas en un matamarcianos cualquiera, podemos ver que la mayoría de esas naves siguen una tabla de movimiento fija (gracias a lo cual, al reiniciar el juego varias docenas de veces, ya sabemos por donde van a salir y los freimos apenas aparecen), en cambio en otros juegos el movimiento parece totalmente aleatorio y en otros el programa genera las posiciones donde pondrá el objeto mediante procedimientos que se encargan de ello.

De las distintas maneras de hacer todo esto, vamos a hablar en esta parte y como el movimiento se demuestra andando, vamos al programa.

BOLA es un corto ejemplo de varios tipos de movimientos. Hay tres objetos en pantalla, LOGO, MONEDA y BOLA. El LOGO sigue una tabla de principio a fin y a la inversa,la BOLA sigue una tabla para su coordenada Y e incrementa o decrementa la coordenada X mediante el programa con unas variables mientras que la MONEDA sigue un movimiento fijo en diagonal hasta chocar en los extremos de la pantalla.

Remitiendonos al listado, vemos en primer lugar una macro que se emplea para acortar el listado evitando teclear una instrucción numerosas veces. El programa empieza en

PRG: MOV AX,13h (19 decimal)

.. con lo que se pone el modo gráfico 320x200, después carga la paleta de colores y los gráficos de la 'demo' BALL y se entra al bucle principal llamado MAIN.

La idea es simple, se imprimen los distintos objetos, se hace una pausa para que la imagen permanezca en el fósforo el tiempo necesario y evitar el parpadéo, se borra la bola por ser un objeto tipo SPRITE, se calcula la próxima posición de los objetos y tras comprobar si se pulsa alguna tecla se reinicia el bucle si no se pulsó o sale al DOS si se hizo.



Lo primero que hacemos es imprimir la MONEDA porque queremos que pase tras el LOGO esto lo hace el procedimiento IMP_MON. Simplemente calcula la dirección donde se imprimirá tomando sus coordenadas de pixel de la esquina superior izquierda y hace un movimiento del bloque gráfico a la pantalla. El gráfico de la MONEDA está en el archivo de gráficos BOLITAS.RAW en la línea 25 col 146 (25*320 +146= 8146). La MONEDA no necesita ser borrada porque tiene una línea vacía (en negro) a su alrededor que al imprimirla borra lo que hubiera debajo, es decir a sí misma.

El procedimiento para mover la MONEDA es MONE_MOV, está explicado en el listado y su filosofía es sencilla. Se tiene un número para cada coordenada que se suma a la línea o la columna en cada caso, este número puede ser 1 ó -1, si es 1, al sumar se incrementa la coordenada y si es -1, con la misma operación de suma, lo que hace es restar. Para que valga 1 ó -1 se comprueba si se han alcanzado los límites de la pantalla y si es así, se invierte ese numero via NEG y se guarda.

Tras la MONEDA va el LOGO, se imprime mediante BLUE_LOGO y el sistema es idéntico al de la MONEDA, calcula la posición en pantalla partiendo de X e Y y mueve el gráfico a pantalla.

El sistema de moverlo es lo que cambia. Consiste en tener un contador [LOGO_C] que sirve de índice a una tabla y que dependiendo de una variable [LOGO_F] que puede ser 1 ó -1 , se incrementa o decrementa para pasar al siguiente elemento de la tabla. Entonces se comprueba que no se superen los límites de la tabla y si es así se invierte la variable y se guardan todas tras las modificaciones para ser usadas por el proc. de impresión.

Y lo último, dado que queremos que pase por encima de los otros objetos, es la BOLA. IMP_BOLA primero calcula la dIrección en pantalla del modo habitual y después mirando [COLOR] apunta el SI al inicio de la BOLA correspondiente, y comienza la impresión. Como no queremos que el rectángulo en el que está la BOLA, se vea borrando lo que haya bajo la BOLA en pantalla hemos de comprobar byte a byte si se debe o no imprimir cada punto del gráfico Existen otros modos de imprimir SPRITES se tratarán en otro momento pero para este ejemplo, este es perfecto. Se repite la misma operación 3 veces para acelerar el proceso, si hubiera más gráficos o fueran de mayor tamaño, este número sería muy superior (el tiempo es crítico cuando se hacen las cosas en pantalla directamente).

BOLA_MOV es una mezcla de los dos métodos de movimiento anteriores. El movimiento a lo largo del eje X es como en la MONEDA solo que algo diferente, suma a la coordenada X un 1 ó -1 que es el contenido de [SI+2] y si se alcanza el lado izquierdo o derecho de la pantalla se invierte [SI+2] mientras que el movimiento a lo largo de Y se realiza como en el LOGO mediante un contador [CONT] que sirve de índice a una tabla [T_BOTE].

Tras imprimir los 3 objetos, hay que dejar que permanezcan en pantalla el tiempo suficiente para que la retina los capte, un método ideal es esperar una señal de barrido del monitor, esto lo hace el procedimiento SINCRO y se explicará en la sección dedicada a la VGA.

Al recibir esa señal es el momento idóneo para borrar la bola, efectuar los cálculos y reiniciar el ciclo pues de esa manera evitamos el parpadeo de los objetos en pantalla. Haz una prueba y elimina la llamada a SINCRO verás a que me refiero.

A parte de eso, en el listado hay otros procedimientos misceláneos que no tienen que ver con el tema como es la carga de los ficheros y que están sobradamente explicados en el mismo listado.

Con esto termina la explicación de esta sencilla rutina de movimiento. Espero que aprendas mucho de ella.

FRAN94



Forma en que se guardan los gráficos en disco.

Listado Bola.asm

```
                TITLE           BOLA

.386                                    ;Se van a usar registros de 32 bits

CODIGO  SEGMENT USE16
        ASSUME  CS:CODIGO, DS:CODIGO
        ORG     100h                    ;Origen necesario en .COM

;Macro para ahorro de tecleo y lineas en el listado fuente
;MOVER pone 'n' n£mero de veces la instrucci¢n MOVSD

mover   MACRO n
        REPT  n
        MOVSD
        ENDM
        ENDM

;:::::::::::::::::::::::::::::::::::CODIGO:::::::::::::::::::::::::::::::::::::

PRG:    MOV     AX,13h
        INT     10h                     ;Poner el modo  320x200 256 colores
        CALL    CARGA_GRAF              ;Carga pantalla y paleta y pone esta
        MOV     AX,0A000h               ;Segmento de memoria de la VGA en ES
        MOV     ES,AX

;:::::::::::::::::::::::::::::BUCLE PRINCIPAL::::::::::::::::::::::::::::::::

MAIN:   CALL    IMP_MON                 ;Imprime moneda

        CALL    BLUE_LOGO               ;Imprime el LOGO

        CALL    IMP_BOLA                ;Imprime la bola en SPRITE

        CALL    SINCRO                  ;Pausa de sincronia

        CALL    BORRA                   ;Borrado de bola

;:::::::::::::::::::::::::::CALCULO DE COORDENADAS:::::::::::::::::::::::::::

        CALL    LOGO_MOV                ;Calcular nuevas X e Y para el LOGO

        CALL    BOLA_MOV                ;   "       "       "       " la bola

        CALL    MONE_MOV                ;  para la MONEDA

;::::::::::::::::::::::::::::::: ¨SALIR? ::::::::::::::::::::::::::::::::::::

        MOV     AH,1                    ;Leer el teclado en busca de
        INT     16h                     ; alguna tecla pulsada
        JNZ     SALIR
        JMP     MAIN                    ;Repetir proceso si no pulsc

SALIR:  MOV     AX,3                    ;Poner en modo texto
        INT     10h

DOS:    MOV     AH,4Ch                  ;Salir al DOS
        INT     21h

;:::::::::::::::::::::::::::::IMPRIMIR BOLA:::::::::::::::::::::::::::::::::::
;Imprime en ES:DI la bola tipo SPRITE (sin borrar todo el rect ngulo que
; define la bola en el gr fico que pueda haber debajo)
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

IMP_BOLA        PROC
        MOV     DI,[X]                  ;Del mismo modo que con el LOGO
        MOV     AX,[Y]                  ; se calcula la direcci¢n en pantalla
        MOV     DX,320                  ; de la bola
        MUL     DX
        ADD     DI,AX                   ;DI= posici¢n en pantalla
        MOV     SI,OFFSET BLANCA        ;SI = Dir. de 1¦ bola
        MOV     AX,[COLOR]              ;N§ de bola
        CMP     AX,6                    ;¨ Ultima bola ?
        JNE     ETIQ1                   ; No, seguir
        SUB     AX,AX
        MOV     [COLOR],AX              ;Poner a 0 = primera bola
ETIQ1:  SHL     AX,1                    ;N§ de bola * 2 para apuntar a la
        ADD     SI,AX                   ; palabra de la bola correpondiente
        MOV     SI,[SI]

        MOV     CH,23                   ;Alto en lineas del gr fico
BUC_ALTO:
        MOV     DX,320                  ;Valor a sumar para linea siguiente
        PUSH    SI                      ;Preservar origen de gr fico y dir. en
        PUSH    DI                      ; pantalla para c lculo de linea sig.
        MOV     CL,9                    ;Ancho/3
                                        ;Se imprime 3 puntos (pixels) cada vez
                                        ; por ser 3 * 9 = 27 que es el ancho
                                        ; del gr fico de la bola
BUC_ANCHO: LODSW                        ;Coger dos puntos del gr fico
        OR      AL,AL
        JZ      S_1                     ;Si el byte es 0, no imprime
        MOV     ES:[DI],AL              ;Imprime un punto
S_1:    INC     DI                      ;Pasa al punto a la derecha
        OR      AH,AH                   ;Misma comprobaci¢n
        JZ      S_2
        MOV     ES:[DI],AH
S_2:    INC     DI
        LODSB                           ;Coge solo un punto
        OR      AL,AL                   ;Comprueba si 0
        JZ      S_3
        MOV     ES:[DI],AL              ; e imprime si no lo es
S_3:    INC     DI
        DEC     CL                      ;Decrementa contador de ancho
        JNZ     BUC_ANCHO               ;Si no cero, repite bucle
        POP     DI                      ;Saca direcci¢n donde imprime bola
        POP     SI                      ;Saca direcci¢n de inicio de la bola
        ADD     DI,DX                   ;Pasa a la linea inferior en pantalla
        ADD     SI,DX                   ; y del gr fico
        DEC     CH                      ;Dec. contador de alto
        JNZ     BUC_ALTO                ;Repite tantas veces como n£mero de
        RET                             ; lineas tiene la bola
IMP_BOLA        ENDP

;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; C lculo de la coordenadas Y de la bola seg£n la tabla T_BOTE (y CONT y FLAG)
; T_BOTE es la tabla de botar. Tambien calcula X como en el mov. de MONEDA
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

BOLA_MOV        PROC
        MOV     AX,[CONT]               ;Coge el CONTador
        MOV     SI,OFFSET T_BOTE        ;Apunta al inicio de la tabla
        ADD     SI,AX                   ;[SI] = valor a sacar de tabla
        LODSB                           ;Lo saca en AL
        LEA     SI,Y                    ;Apunta SI a la posici¢n donde est
                                        ; la coord. Y de la bola
        MOV     AH,[FLAG]               ;Comprueba su 'FLAG de direcci¢n'
        OR      AH,AH                   ; si FLAG = 1
        JNZ     B_RESTA                 ; debe restar a Y el [SI]
        ADD     [SI],AL                 ; si FLAG = 0, suma
        INC     [CONT]                  ;Incrementa la variable contador
        MOV     AX,60
        CMP     [CONT],AX               ;Comprueba que no exceda el m ximo
        JNZ     EXIT1
        MOV     AL,1                    ;Si excede, invertir FLAG para resta
        MOV     [FLAG],AL               ; la pr¢xima vez
        JMP     SHORT EXIT1             ;Salta a punto com£n
B_RESTA: SUB    [SI],AL                 ;Si FLAG fue = 1, resta [T_BOTE] de Y
        DEC     [CONT]                  ; y dec. el contador
        MOV     AX,[CONT]
        OR      AX,AX                   ;Comprobar si llegc a 0
        JNE     EXIT1
        MOV     AL,0                    ;Si fue asi, invertir FLAG. Pr¢xima
        MOV     [FLAG],AL               ; vez sumar
EXIT1:  LEA     SI,X                    ;Apunta SI a columna
        MOV     AX,[SI]                 ;Palabra de 0 a 293 en AX
        MOV     DX,AX
        OR      AX,AX                   ;¨Izquierda de pantalla?
        JZ      COLU_0
        CMP     AX,293
        JNZ     INC_COL                 ;¨Derecha de pantalla?
COLU_0: INC     [COLOR]                 ;Inicio o fin, cambiar color de bola
        MOV     AX,[SI+2]               ;Invertir incremento en X
        NEG     AX                      ;Si era 1 pasa a -1 y viceversa
        MOV     [SI+2],AX               ;Guarda y la pr¢xima vez ir  al reves
INC_COL: MOV    AX,[SI+2]
        ADD     AX,DX                   ;Suma a la columna un 1 c un -1
        MOV     [SI],AX                 ;Guarda columna
        RET
BOLA_MOV        ENDP

;::::::::::::::::::::::::::::::::::
; Procedimiento de mover la moneda ::::::::::::::::::::::::::::::::::::::::::::
;::::::::::::::::::::::::::::::::::

MONE_MOV        PROC
        LEA     SI,MONEDA_Y             ;Esto cambia las coordenadas
        MOV     AL,[SI]                 ;Coge Y
        MOV     AH,AL                   ;Guarda
        OR      AL,AL                   ;Si es 0 cambiar incremento
        JZ      LIN_0
        CMP     AL,160                  ;¨Es la £ltima linea
        JNZ     SUMA_L
LIN_0:  MOV     AL,[SI+2]               ;Coje incremento
        NEG     AL                      ;Pone en decremento
        MOV     [SI+2],AL               ;Guarda
SUMA_L: MOV     AL,[SI+2]               ;Coje incremento o decremento
        ADD     AL,AH                   ;Suma a LINea
        MOV     [SI],AL                 ;Pone en LINea
        LEA     SI,MONEDA_X             ;Apunta a columna
        MOV     AX,[SI]                 ;Palabra de 0 a 279 (320-36)
        MOV     DX,AX
        OR      AX,AX
        JZ      COL_0                   ; ¨0?
        CMP     AX,279                  ;Sale de la pantalla ?
        JNZ     SUMA_C
COL_0:  MOV     AX,[SI+2]               ;Invertir
        NEG     AX
        MOV     [SI+2],AX
SUMA_C: MOV     AX,[SI+2]
        ADD     AX,DX
        MOV     [SI],AX
        RET
MONE_MOV        ENDP

;:::::::::::::::::::::::::::::::::::BORRADO:::::::::::::::::::::::::::::::::::
;Esta parte borra la bola en la pantalla, simplemente toma los mismos datos
; que al imprimir y llena con ceros
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

BORRA:  MOV     DI,[X]
        MOV     AX,[Y]
        MOV     DX,320
        MUL     DX
        ADD     DI,AX
        MOV     CX,23                   ;Alto
        SUB     EAX,EAX                 ;Rellena con 0
        MOV     DX,320
BU_ALTO: PUSH   SI
        PUSH    DI
        STOSD                           ;Primera rebanada
        STOSD
        STOSD
        STOSD
        STOSD
        STOSD
        STOSW                           ;Linea borrada
        POP     DI
        POP     SI
        ADD     DI,DX                   ;Linea inf. pantalla
        LOOP    BU_ALTO
        RET

;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;Este procedimiento imprime la moneda
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

IMP_MON PROC
        MOV     DI,[MONEDA_X]           ;C lculo de coordenadas en pantalla
        MOV     AX,[MONEDA_Y]
        MOV     DX,320
```



Listado Bola.asm

```
        PUSH    DX
        MUL     DX
        ADD     DI,AX                       ;DI apunta al destino
        POP     DX
        MOV     SI,MONEDA                   ;SI apunta al origen del gr fico
        MOV     CX,36                       ;Alto del gr fico
B_ALTO: PUSH    SI
        PUSH    DI

B_ANCHO: mover  10                          ;10 MOVSD * 4 = 40 bytes movidos

        MOVSW                               ; m s 2 del MOVSW = 42 = ANCHO
        POP     DI
        POP     SI
        ADD     DI,DX
        ADD     SI,DX                       ;Siguiente rebanada
        LOOP    B_ALTO                      ;Repetir alto
        RET
IMP_MON ENDP

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; Realiza los c lculos del movimiento del logo seg£n el CONTADOR , el FLAG y
; la tabla de INERcia
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

LOGO_MOV        PROC
        MOV     AX,[LOGO_C]                 ;Saca de la tabla el valor apuntado
        MOV     SI,OFFSET T_INER            ; por el contador
        ADD     SI,AX
        LODSB
        LEA     SI,LOGO_Y
        MOV     AH,[LOGO_F]                 ;Mira si hay que sumar o restar ese
        OR      AH,AH                       ; valor
        JNZ     RESTA
        ADD     [SI],AL
        INC     [LOGO_C]                    ;Incrementa contador y comprueba si
        MOV     AX,93                       ; llega al final de la tabla
        CMP     [LOGO_C],AX
        JNZ     EXIT
        MOV     AL,1                        ;Si es asi, activa [flag] de resta
        MOV     [LOGO_F],AL
        JMP     SHORT EXIT
RESTA:  SUB     [SI],AL
        DEC     [LOGO_C]                    ;Dec. contador para ver si lleg¢ al
        MOV     AX,[LOGO_C]                 ; principio de la tabla
        OR      AX,AX
        JNE     EXIT
        MOV     AL,0                        ;Activa [flag] para suma
        MOV     [LOGO_F],AL
EXIT:   RET
LOGO_MOV        ENDP

;:::::::::::::::::::::::::::::::::::SINCRO:::::::::::::::::::::::::::::::::::::
;Esperar la se¤al de retrazo

SINCRO: MOV     DX,03DAh
SINCO:  IN      AL,DX
        TEST    AL,8
        JNE     SINCO
SINC1:  IN      AL,DX
        TEST    AL,8
        JE      SINC1
        RET

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; Imprime el LOGO en la linea colunma calculada por LOGO_MOV
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

BLUE_LOGO       PROC
        MOV     DI,[LOGO_X]                 ;DI = Columna
        MOV     AX,[LOGO_Y]                 ;AX = Linea
        MOV     DX,320                      ;N§ de bytes por linea
        MUL     DX                          ;Linea * 320 + Columna = Posici¢n en
                                            ; pantalla
        ADD     DI,AX                       ;ES:DI = Segmento:desplazamiento
        MOV     SI,LOGO                     ;Esquina sup. izq. del objeto a imprim.
        MOV     CX,76                       ;Alto
        MOV     AX,320                      ;Valor a sumar para linea inferior

L_ALTO: PUSH    SI
        PUSH    DI

BUCLE:  mover   36                          ;36 MOVSD =144 bytes movidos = ancho

        POP     DI
        POP     SI
        ADD     DI,AX
        ADD     SI,AX                       ;Siguiente rebanada
        LOOP    L_ALTO
        RET

BLUE_LOGO       ENDP

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;Procedimientos de carga del fichero de paleta y de los gr ficos
;Tras cargar la paleta de colores llama a PALETA que la pone
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

CARGA_GRAF      PROC
        MOV     AX,3D00h                    ;Abrir fichero
        MOV     DX,OFFSET [PALETB]          ; DS:DX nombre de archivo
        INT     21h
        JNC     BIEN
        MOV     DX,OFFSET [ERR1]            ;No se pudo abrir, error
ERROR:  MOV     AH,9                        ;Muestra error y sale al DOS
        INT     21h
        JMP     DOS
BIEN:   MOV     HANDLE,AX                   ;Guarda HANDLE para otros usos
CARGA:  MOV     AH,3Fh                      ;Cargar fichero
        MOV     BX,HANDLE
        MOV     CX,776                      ;Longitud del fichero de paleta
        MOV     DX,OFFSET BUFFER            ; en buffer tras programa
        INT     21h
        JNC     CERRAR                      ;Si bien, seguir
        MOV     DX,OFFSET [ERR2]            ; si no, salir con error
        JMP     SHORT ERROR
CERRAR: MOV     AH,3Eh                      ; cerrar fichero
        MOV     BX,HANDLE
        INT     21h

        CALL    PALETA                      ;Poner la paleta

        MOV     AX,3D00h                    ;Abrir fichero
        MOV     DX,OFFSET [FICHERO]         ;Fichero con los gr ficos
        INT     21h
        JNC     BIEN1                       ;Fichero abierto
        MOV     DX,OFFSET [ERR1]
        JMP     ERROR                       ;Error, salir
BIEN1:  MOV     HANDLE,AX
CARGAR: MOV     AH,3Fh                      ;Cargar fichero
        MOV     BX,HANDLE
        MOV     CX,38400                    ;Longitud de BOLITA.RAW
        MOV     DX,OFFSET BUFFER
        INT     21h
        JNC     CIERRA                      ;Bien, seguir
        MOV     DX,OFFSET [ERR2]            ;Error, salir
        JMP     ERROR
CIERRA: MOV     AH,3Eh                      ;Cerrar fichero
        MOV     BX,HANDLE
        INT     21h
        RET
CARGA_GRAF      ENDP

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;Procedimiento que pone la paleta de colores recien cargada en BUFFER.
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

PALETA: CLD                                 ;Quiero que LODSB incremente el SI
        MOV     SI,offset BUFFER+8          ;Inicio de paleta
        MOV     CX,256                      ;256 registros de paleta
        MOV     DX,03C6h
        MOV     AL,255
        OUT     DX,AL                       ;HE, VGA, que van unos datos
        XOR     BX,BX                       ;Primer registro de paleta
b_out:  MOV     DX,03C8h                    ;Indicar el n§ de registro
        MOV     AL,BL
        INC     BX
        OUT     DX,AL
        INC     DX
        LODSB
        SHR     AL,2
        OUT     DX,AL                       ;Primer componente del color
        LODSB
        SHR     AL,2
        OUT     DX,AL                       ;Segundo
        LODSB
        SHR     AL,2
        OUT     DX,AL                       ;Tercero
        LOOP    b_out                       ;Repetir 256 veces
        RET

;::::::::::::::::::::::::::::::::::::DATOS:::::::::::::::::::::::::::::::::::::

FICHERO DB      "BOLITA.RAW",0
PALETB  DB      "6x32.COL",0
HANDLE  DW      0
ERR1    DB      "No se puede abrir fichero $"
ERR2    DB      "Fallo de carga $"

;Datos de la BOLA
COLOR   DW      0                           ;N§ de bola
X       DW      1,1                         ;Coordenada X e incremento
Y       DW      88,1                        ;Coordenada Y e incremento
CONT    DW      1                           ;Contador indice a tabla
FLAG    DB      0                           ;Flag usado para suma/resta

MONEDA_X  DW      100,1                     ;Datos de MONEDA
MONEDA_Y  DW      30
          DB      1

LOGO_X  DW      90,1                        ;Datos del LOGO
LOGO_Y  DW      20,1
LOGO_C  DW      1
LOGO_F  DB      0

LOGO    DW      BUFFER+7360                 ;Offset del gr fico del logo en el
                                            ; fichero
BLANCA  DW      BUFFER                      ;Idem de la bola blanca
ROJA    DW      BUFFER+29                   ; "   "  "   "   roja   etc.
VERDE   DW      BUFFER+57
AZUL    DW      BUFFER+86
AMARIL  DW      BUFFER+112
GRIS    DW      BUFFER+140

MONEDA  DW      BUFFER+8146

; T_BOTE        60 DATOS , Para la bola
; T_INER        93 DATOS , Para el logo

T_BOTE  DB      0,0,1,0,0,1,0,0,1,0,0,1,1,0,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1
        DB      1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3,3,3,4,4,4

T_INER  DB      0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,1,0,1,1,0,1,1,0,1,1,1,1,1
        DB      1,1,1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,3,3,2,2,2,2,2,1,1,1,1,1,1,1
        DB      1,1,1,1,1,0,1,1,0,1,1,0,1,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0

BUFFER  =  $

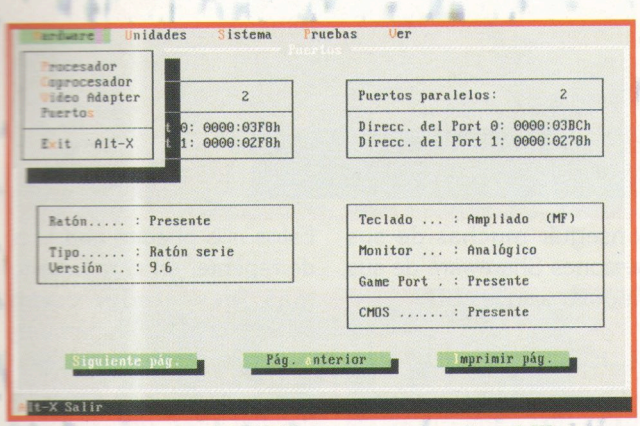
CODIGO  ENDS
        END PRG
```



El Libro del mes

El gran libro del 486

Editorial Marcombo, 555 páginas y disquete



Pantalla de PCINFO que acompaña al libro

Conocer bien el ordenador que manejamos es una asignatura importantísima cuyo desconocimiento llevará al programador a un choque irrevocable con un muro difícil de salvar a la hora de realizar sus aplicaciones. Por ello, debemos hacernos de buenos libros que nos expliquen las interioridades del sistema que manejamos y que nos ayuden a salir de esos atolladeros en los que nos metemos a veces y de los que no sabemos escapar por más fuerte que gritemos pidiendo auxilio.

Son muchos los libros que salen al mercado dedicados a este fin, pero no debemos confundirnos. Muchos adolecen de una explicación simplificada y dirigida a la persona que normalmente los compra, es decir el programador que como la mayoría de los que estáis leyendo estas líneas, no tiene mucha o tanta idea sobre lo que se está tratando. Estos mamotretos recopilan información en forma de tablas y listados que sólo hablan de números y letras que poco o nada dicen al que quiere aprender.

No ocurre así con el libro que he escogido para recomendaros este mes; El gran libro del 486.

Como sus mismos autores dicen, se decidieron a escribir esta obra de por dos razones principales. La primera la he descrito ya en mi comentario de los párrafos anteriores, es decir, falta de ejemplos y explicaciones prácticas sobre como funciona la plataforma PC. La segunda sería la poca documentación que traen los sistemas cuando los adquirimos. Tras gastarnos un buen dinero en un sistema de computación, abrimos la caja y cual no será la sorpresa al ver que tan sólo unas pocas páginas con información confusa acompañan al corcho del enbalaje. Principalmente si nos compramos un clónico de PC, que entonces tiene, como dice el dicho, menos papeles que "una yegua robada".

Gracias a Dios, algunas personas se dedican a recopilar información y ponerla en blanco y negro de una forma clara y dirigida a todos. Bueno, para ser totalmente francos, no todo el mundo podría entender la información que llena las páginas de este libro, pero si os aseguro que todo aquel que tenga unos pocos conocimientos de informática va encontrar un gran apoyo en él y verá que el libro no es en absoluto complicado.

EL GRAN LIBRO DEL 486

Josef Haas y Thomas Jungbluth, autores del libro, demuestran haber tenido un gran cuidado a la hora de escoger los temas y desarrollarlos. Eso se nota, principalmente, por la forma en que las diferentes materias han sido tratadas en orden ascendente de dificultad y por la manera en que se han separado todos los aspectos que conciernen al procesador 486 y a aquellos sistemas que se basan en él.

COMETIDO Y PARTES

El libro pretende ilustrar y enseñar los aspectos principales de construcción y funcionamiento de los sistemas basados en el microprocesador Intel 486, aunque también encontramos información muy valiosa para cualquiera que sea el procesador que lleve incorporado nuestro PC.

Alguna de la información que encontramos en sus páginas no la hemos visto en ningún otro libro parecido, lo que lo hace bastante completo en cuanto a este aspecto. Esta, por ejemplo, nos habla de las diferentes formas de las carcasas del PC, tipos de placas madre (motherboards), ergonomía de los sistemas informáticos, consejos de compra, etc.

Tras esta introducción y después de haber echo una breve pero interesante historia, sobre el PC y su nacimiento, Haas y Jungbluth se adentran en la materia sustanciosa del libro en el que se describen exaustiva pero cautelosamente todos los principales componentes de un PC.

También, hay un rincón para los diferentes sistemas operativos que encontramos para el 486 y por defecto para todos los otros micros de la familia Intel (8086, 286, 386), por lo que ya os digo que el nombre no debe asustar a los que no tengan un ordenador con el 486 dentro.

Finalmente, se habla un poco sobre Windows y sus diferentes componentes y configuraciones.

También se describen muchos de los shells o entornos del usuario que están disponibles para el PC, como el GeoWorks, GEM, etc.

En el apartado "Programas para el 486" encontramos aquellos avances que ha supuesto una mejor arquitectura interna de este microprocesador, debido al mejor y más eficaz manejo que éste hace de la memoria y por la inclusión del coprocesador matemático. Se explican aquí como se usan todas las posibilidades que nos ofrece el MSDOS 5.0 para el aprovechamiento de este fantástico microprocesador.

Otro programa capaz de manejar la memoria expandida y extendida del PC es 386max el cual es revisado en uno de los apartados del libro.

Por último, el disquete que acompaña el libro. Los programas, y según la propia editorial DATA BEQUER, han sido escogidos minuciosamente por los autores para acompañar la explicaciones del libro y para demostrar algunas características especiales del héroe del libro; el microprocesador 486. El disquete tiene un programa llamado PCINFO que nos informa de todos los aspectos principales del sistema donde los hacemos funcionar. Tiene varios menús como podemos ver en la imágen adjunta. Así tenemos "hardware", "unidades", "sistema", "pruebas" y "ver" y por último el listado del programa PCINFO hecho en PASCAL.

El libro finaliza con unos completísimos apéndices de información variada y estructurada así como un glosario de términos utilizados en la obra.

NUESTRA VALORACION

El libro está bien hecho y desarrollado. Lo vemos un poco falto de información profunda para programadores más avanzados, pero por el contrario constituye un mágnifico manual para los que quieren conocer un ordenador por dentro y están empezando a dar sus pasos por el mundo del PC. Otra recomendación de Código Juego para sus programadores.

Wolf-3D, Spears of Destiny, In Extremis, Corridor 7, simuladores como Comanche Maximum Overkill, Strike Eagle III... Todos ellos tienen una cosa en común: su impresionante perspectiva en tres dimensiones. Pero Doom da un palmetazo a todo esto para situarse en cabeza de los juegos de arcade y acción.



Compañía ID Software

Colaborador

Raul Reales González

UN POCO DE HISTORIA:

Hace ya algunos años, cayó en nuestras manos la versión Shareware de lo que hoy es considerado un clásico. Nos estamos refiriendo a Wolfenstein-3D, que se incluyó en los discos una conocida revista. Algo que causó una gran sensación, fueron, y todavía siguen siendo, sus rutinas de movimiento en tres dimensiones. Consistían en una vista en primera persona de todo lo que a uno le rodea, a gran velocidad. Todo ello sin olvidar la calidad de sus gráficos, que le dotaban de un gran realismo.

Era el único juego conocido hasta el momento -por lo menos en España- que reunía estas características tan sensacionales. Desde luego suponían un adelanto en lo que se refiere a los videojuegos.

Pero eso no se quedó así. ID Software, la compañía que había desarrollado el producto, decidió no dormirse en los laureles, y tomó la decisión de seguir adelante con este tipo de arcade. Al cabo del tiempo, consiguieron lanzar la segunda parte de Wolf-3D, conocido como Spears of Destiny, que realmente lo único que reunía eran nuevas misiones y más enemigos, con lo cual sólo dejaron satisfechos a los que deseaban seguir con la acción, sin importarles mucho el apartado referente a las mejoras.

Por las mismas fechas aparecía un simulador llamado Comanche, que utilizaba una técnica en tres dimensiones denominada *Voxel Space*. Ella conseguía parecerse mucho a las rutinas del Wolf-3D. Eso sí, la calidad de los gráficos era muy alta, con el inconveniente de una gran lentitud en equipos 386 de gama baja debido a su alto nivel de detalle, el cual, afortunadamente se podía disminuir a gusto del usuario. A partir de aquí, y con la aparición de más simuladores conforme pasaba el tiempo -F-15 Strike Eagle III, Strike Comander, Subwars 2050- hacían dudar de la verdadera procedencia de esta técnica.

Realmente, la huella de Wolf-3D, se hace notar debido a unos rumores de la versión Enhanced (mejorada) de la misma. Esto puede decir mucho a su favor, para así volver a ponernos al control de ese personaje mata-nazis y disfrutar de todas las sorpresas con todas esas mejoras.

Lamentablemente, la mayoría de los lanzamientos de ID Software no son comercializados en España, pero se pueden conseguir muchas de sus versiones Shareware de sus productos en las BBS.

EL JUEGO:

A Doom se le puede ver como una parte más de Wolf-3D, pero no es así. Es un programa totalmente nuevo, y reúne unas mejoras de tal calidad, que se debe hablar de él como algo aparte, guardando sólo relación en la rutina, además mejorada. En este juego te mueves por un verdadero mundo tridimensional, no como ocurría en anteriores juegos de ID y otras compañías con desarrollo lineal, pudiendo así saltar, subir y bajar plantas, caer a pozos de ácido, etc. Encarnas a un soldado en una guerra interdimensional cuya misión es, además de exterminar a todo bicho viviente, conseguir encontrar la salida en cada nivel. Para llegar al final debes de rendir al máximo consiguiendo así una buena puntuación. Esto quiere decir que cuantos más enemigos destruyas y más items recojas en el menor tiempo posible, más alto será tu récord.

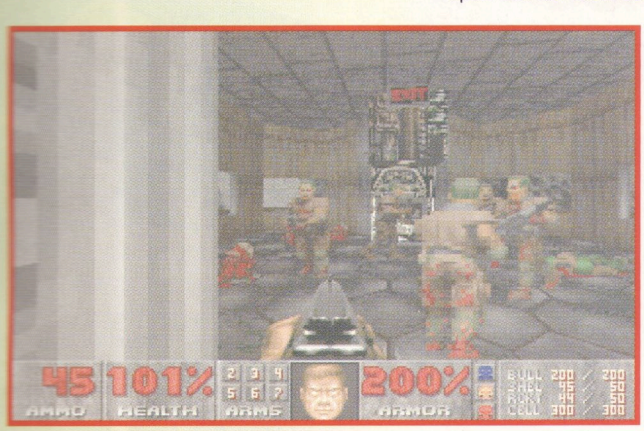
Por supuesto, no todo va a ser un camino de rosas, ya que se nos interpondrán todo tipo de obstáculos que nos harán la vida imposible -demonios, soldados, cabezas, etc.-. Estos nos pueden aparecer de repente, en el momento más inesperado, como puede ser al abrir una puerta, cruzar una esquina, etc., dependiendo sólo del nivel de dificultad que se haya elegido. Otra novedad -muy buena- es la incorporación del auto-mapa, que se activa con la tecla de tabulación. Una vez activado, la perspectiva pasa a ser de dos dimensiones, donde podemos seguir moviéndonos, pero viendo la estructura del plano que se va descubriendo conforme avanzamos. Cada misión es completamente diferente de la anterior en cuanto a ambientación, haciéndolo más entretenido. En cada una se podrán encontrar los llamados items, los cuales son objetos que proporcionan ventajas a nuestro héroe como pueden ser nuevas armas -rifle, lanzamisiles, cañón de plasma-, vidas extra, inmunidad, y otras más. No podían faltar las zonas secretas, donde se pueden conseguir items más jugosos.

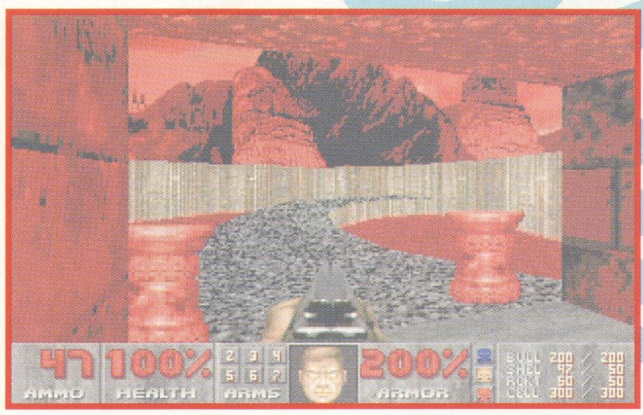
VALORACIÓN:

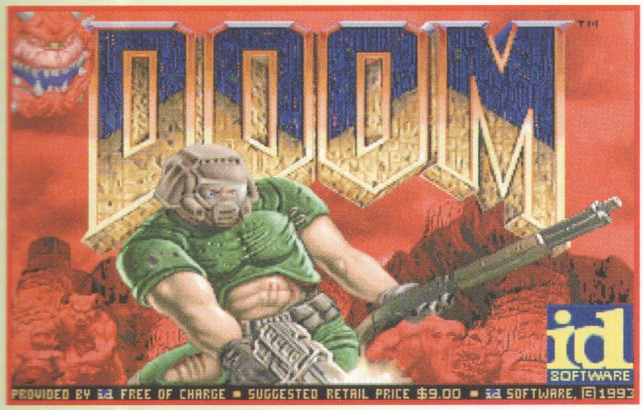
Tanto los gráficos como el sonido -en especial para los poseedores de Sound Blaster- te introducen desde el principio en la atmósfera del juego, que puede hacerte pasar

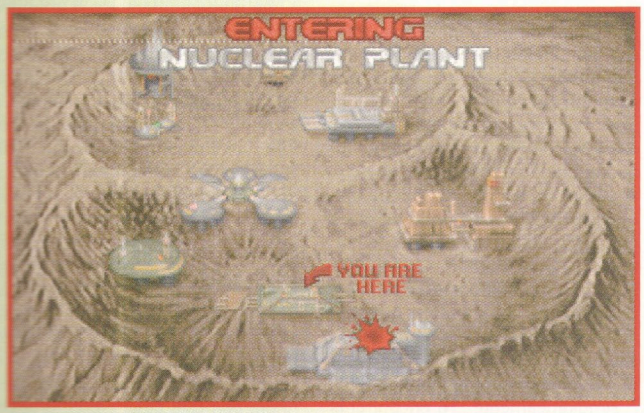
Truco secreto ➡

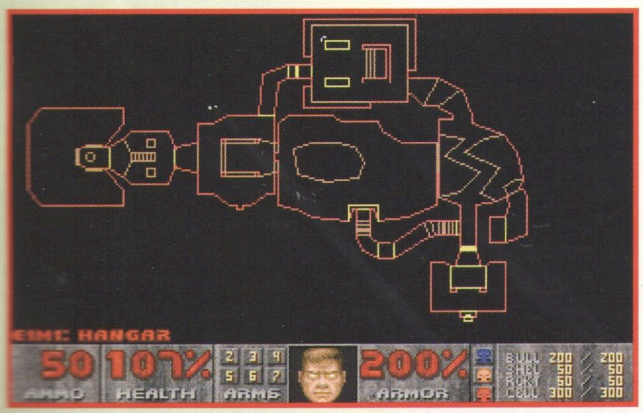
Doom tiene dos combinaciones de teclas que nos facilitarán mucho las cosas. Tecleando IDDQD pasaremos a ser inmunes durante todo el juego. Con IDKFA obtendremos todas las armas, recargaremos la munición y se nos darán las tarjetas de acceso. Esto último se puede repetir cuantas veces desees.











momentos de mucha tensión. A esto contribuye unas extremadas dosis de violencia, que hacen vernos borbotones de sangre y gráficos morbosos con bastante frecuencia -atención a los empalados-, estando siempre presente, haciendo que desde el primer disparo, pidas más y más -intentad controlaros cuando dejéis de jugar, y tened compasión del "perrito de la vecina"-.

En el apartado de la programación, este juego es algo de lo que muchas compañías profesionales deberían tomar ejemplo. El programa en sí, se ejecuta en modo protegido y con un mínimo de cuatro megas de RAM para funcionar correctamente, debido a la cantidad de objetos que necesita almacenar en memoria cuando carga un nivel. Es obvio que al decir modo protegido, es obligatorio tener un 386 ó 486.

Lo que más llama la atención en cuanto a avances, es el denominado Deathmatch. Así pueden jugar hasta un máximo de cuatro jugadores, ya sea por módem o conexión por interface entre ordenadores, siendo cada jugador un soldado independiente que debe completar su misión como cualquiera de los restantes. Esto da lugar a varios modos de juego. Desde luego, es alucinante jugar sabiendo que en el mismo nivel hay otro soldado que también busca la salida y...¿por qué no?, a ti. también.

CONCLUSIÓN:

En general, se puede decir de Doom que es un juego muy completo. Incluye un manual de unas 24 páginas que te documentan extensamente sobre todas las funciones de las teclas -que son muchas-, el argumento, trucos en general y más. A los amantes del arcade puro y de la acción les hará pasar las horas como si fuesen minutos. Su gran cantidad de opciones, como el nivel de detalle -ordenadores no tan potentes- y el de dificultad, tamaño de la pantalla, grabar la situación, mirar la situación en el plano de lo recorrido, etc., hacen que sea un juego difícil de olvidar. Quizás el único problema sea esa odiosa regla de tres potencia/calidad, aunque por esta vez, parece que han querido tener en cuenta al usuario medio, o sea, el 386.

Raúl Reales González

Alberto García

Curso de Pascal

INTRODUCCION

Sin duda, alguna vez has oído hablar de este popular lenguaje, pero, ¿qué puede realmente aportar el Pascal a la programación de juegos? Bien, comencemos desde el principio y veamos como éste evoluciona a lo largo de su historia hasta la actualidad.

Fué a principios de los años setenta cuando un profesor del Instituto Federal de Tecnología de Zurich (Suiza) llamado Niklaus Wirth crea un nuevo lenguaje totalmente estructurado (veremos que es ésto ahora). La idea inicial con la que Wirth crea éste lenguaje es puramente académica, es decir, para mostrar los conceptos básicos de la programación estructurada y como herramienta para la enseñanza de la informática. Evidentemente, hablamos del lenguaje Pascal. Poco a poco éste comienza a extenderse de una manera inusual, hasta establecerse como uno de los lenguajes más usados por los programadores. Más tarde, aparecerían más lenguajes del tipo estructurado como el C, ADA, o el Modula2 (éste último también creado por Wirth). A estas alturas, te estarás preguntando qué es eso de un lenguaje estructurado. Pues bien, los primeros lenguajes iban ejecutando instrucción tras instrucción, y como mucho, poseían una instrucción para saltar de un sitio a otro del programa. Más tarde, a finales de los sesenta, comenzó a surgir una corriente de teóricos informáticos que empezaron a promover los lenguajes estructurados. Uno de los primeros lenguajes de este tipo fue el ALGOL. Lo que distingue al lenguaje estructurado, es que se compone de tres tipos de estructuras fundamentales, con las cuales se pueden construir cualquier tipo de programa. Estas estructuras básicas son:

- Estructura secuencial : El programa se va ejecutando secuencialmente instrucción a instrucción.
- Estructura repetitiva : Repetición de un conjunto de instrucciones mientras/hasta que se cumpla una condición.
- Estructura condicional : Consiste en una bifurcación dependiendo del resultado de evaluar una condición.

No te preocupes si ésto te suena a chino, ya lo entenderás más adelante.

Retomemos la pregunta inicial, ¿qué ofrece el Pascal en el campo de la programación de videojuegos? El lenguaje Pascal, ha evolucionado mucho desde entonces, hoy día se acepta como un estándar el Turbo Pascal, que es un compilador muy potente creado por Borland. Turbo Pascal (TP) tiene un montón de funciones y procedimientos (veremos que es esto en próximos capítulos) para manejo de gráficos y otras cualidades útiles para los juegos, pero si esto fuera lo único que el TPascal nos ofrece, sólo estaríamos hablando de un lenguaje corriente. TPascal, ofrece muchas facilidades para trabajar con la memoria del PC, así como para el manejo de puertos de entrada/salida (E/S) de forma que podrás manejar la tarjeta de video o de sonido directamente. TPascal, ademas tiene la posibilidad de combinarse con el Ensamblador y el C, de forma que podrás escribir el código mas crítico (en cuanto a velocidad) en ensamblador y luego incluirlo como parte del programa pascal, pero ademas podrás usar el ensamblador interno que posee el TPascal, llamado BASM.

En la actualidad, la programación estructurada, está dejando paso a otra metodología de programación, me refiero a la programación orientada a objetos (OOP), sin embargo, TPascal ha sabido adaptarse a su tiempo, y actualmente está totalmente capacitado para la programación orientada a objetos de una manera fácil, sencilla, y a la vez potente.

EL ENTORNO INTEGRADO (GUI)

TPascal, posee un entorno de trabajo, integrando todo lo necesario dentro del editor. Antes de pasar a describir el lenguaje Pascal, tendremos que dar un pequeño repaso al uso del editor y de sus herramientas. Lo primero que tendremos que hacer es cargar el TPascal en memoria, para ello, sitúate en el directorio en el que esté instalado y teclea TURBO seguido de la tecla ENTER. La pantalla que aparecerá ahora, dependerá de la versión de TPascal que tengas, por lo general es aconsejable tener una versión superior a la 5.5, ya que es a partir de ésta versión en la que se dota al TPascal con la capacidad de programación orientada a objetos. Si eres de los que gustan de experimentar y "toquetear" todo, no te reprimas, prueba todas las opciones que quieras, sin miedo, el ordenador está bajo tu control siempre, y lo mejor para trabajar con él es perderle el miedo. Eso si, si cambias alguna opción, ponla como estaba de nuevo cuando termines, sino puedes armar un buen follón (pero nada que no se arregle instalando TPascal de nuevo). El primer menú que encuentras se llama FILE (en la versión inglesa) y posee los comandos básicos para cargar y grabar ficheros ASCII (texto normal), que en nuestro caso serán programas. Para acceder a este menú, pulsa las teclas ALT y F simultáneamente. Te encontrarás con varias opciones que puedes seleccionar moviéndote con los cursores. Selecciona la opción NEW, verás como aparece una ventana en la que aparece el cursor parpadeante. Ahora escribe lo que quieras y seguidamente vuelve al menú y selecciona SAVE AS o WRITE TO dependiendo de la versión que uses. Aparecerá ahora una ventana en la que se nos pide el nombre del programa, pon el que más te guste pero con la extensión PAS, por ejemplo TEXTO.PAS y pulsa ENTER. Ahora nuestro texto está guardado en el disco duro. Vuelve a entrar en el menú y selecciona OPEN. De nuevo aparece una ventana, pero esta vez podemos seleccionar el archivo que queremos leer a través de los cursores o el ratón. Si seleccionas el que grabamos antes, lo verás aparecer en una ventana, listo para volver a ser modificado o ampliado. Para salir del editor de TPascal, ve al menú FILE y selecciona la opción EXIT. Como ves no es nada difícil trabajar con los ficheros, ahora veremos como editar los textos (programas). Esto es algo que puede parecer trivial, pero puede ayudar muchísimo a completar programas de manera rápida y más eficiente. Para empezar carga el texto que escribistes antes, si no es muy grande, completalo hasta que tenga unas cuantas líneas.

La primera operación que vamos a realizar es copiar bloques de texto, como verás es muy fácil y útil si tienes que repetir un trozo de código (programa) varias veces. Si tienes la versión 5.5 o anterior, el editor usa los comandos estándar de WordStar, así que colocate al principio del texto que quieras copiar (solo en versiones 5.5 o anterior) y pulsa la tecla CONTROL y K a la vez, seguidamente pulsa la letra B. Ahora muevete hasta el final del trozo de texto que quieras copiar y pulsa CONTROL y K (CTRL+K), después pulsa la letra K y verás como se selecciona el texto que tú querías. Ahora con este bloque podrás hacer lo quieras, por ejemplo, mueve el cursor hasta el final del texto y pulsa CTRL+K y seguidamente C, te darás cuenta de como se hace una copia del texto seleccionado. Otras operaciones que puedes hacer es:

CTRL+K - C : Copiar bloque.
CTRL+K - Y : Borrar bloque.
CTRL+K - V : Mover Bloque.
CTRL+K - H : Quita las marcas de bloque.

Si usas la versión 6.0 o posterior la cosa es mucho mas fácil, de hecho usa los mismos comandos que el editor del MS-DOS (EDIT). Para seleccionar un bloque, simplemente colocate al principio de él, pulsa SHIFT (una tecla con una flecha hacia arriba) y mientras la mantienes

pulsada, mueve el cursor hasta el final del bloque. Ahora ya está marcado y sólo tienes que ir al menú EDIT (ALT+E) y seleccionar la opción que más te convenga, de todas formas te aconsejo que te aprendas las teclas "atajo" para los comandos de edición, éstas aparecen al lado derecho de cada opción del menú.

NUESTRO PRIMER PROGRAMA

Ya es el momento de introducir nuestro primer programa, así que elige NEW en el menú FILE y copia LITERALMENTE el programa que aparece en la figura 1. Cuando lo hayas tecleado (no te preocupes si no lo entiendes), ve al menú RUN (ALT+R) y pulsa la opción RUN. En pantalla habrá aparecido una ventana y antes de que te haya dado tiempo a ver lo que contenía, la pantalla habrá hecho un guiño y habrá aparecido de nuevo la pantalla de edición del TPascal (si no ha ocurrido así repasa el programa). Bien en ese segundo el TPascal ha compilado y ejecutado nuestro programa. Como ves TPascal es un verdadero rayo compilando. La misión de nuestro programa era escribir en la pantalla la frase "HOLA MUNDO." (si sabes algo de inglés, seguramente ya lo imaginabas), pero sin embargo, no ha salido. Bien Pulsa las teclas ALT+F5 y observa la pantalla que aparece, verás que nuestro programa ha hecho lo que queríamos que hiciera. No está mal para empezar ¿no? Lo que has hecho es pasar a la pantalla de ejecución, ya que una vez que el programa ha sido ejecutado, TPascal regresa a la pantalla de edición. Ahora seguramente te gustará cambiar lo de "HOLA MUNDO" para que aparezca tu nombre o cualquier otra cosa, hazlo, el programa es tuyo y puedes modificarlo a tu antojo, pero mantén lo que quieras escribir entre las comillas. Ahora pulsa de nuevo la opción RUN o directamente CTRL+F9 y seguidamente ALT+F5, ahí estará lo que le habíamos dicho al TPascal que escribiera. Antes de pasar a describir línea a línea lo que

```
PROGRAM   hola (input,output);

BEGIN
      writeln('Hola Mundo.');
END.
```

Figura 1

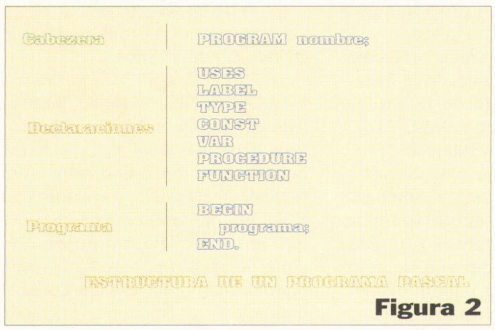


Figura 2

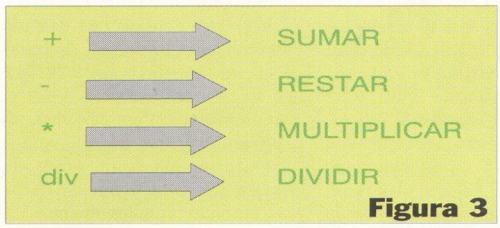


Figura 3

```
PROGRAM triangulo;

USES crt;
VAR
   base : integer;
   altura : integer;
   resultado : integer;

BEGIN
   clrscr;                          (* Borra la pantalla *)
   base := 20;
   altura := 10;
   resultado := (base * altura) div 2;
   writeln ('El area del triangulo es ', resultado);
END.
```

Figura 4

PROGRAM nombre_del_programa;

ESTRUCTURA DE UN PROGRAMA EN TPASCAL.

COMO FUNCIONA EL PROGRAMA

GRAPH. La sintaxis general de éste bloque de declaraciones es:
USES unidad1,unidad2,...,unidadN;

En el ejemplo que poníamos de los gráficos, la línea quedaría de la siguiente manera:

USES graph;

La unidad más importante, es la unidad CRT, ya que posee todas las instrucciones relacionadas con la entrada/salida en modo texto en pantalla.

El siguiente bloque es el bloque de etiquetas. Sinceramente te aconsejo que lo olvides, ya que sirve para poder usar la instrucción GOTO que da un salto de un lugar a otro del programa, y como dijimos antes, ésto es lo que la programación estructurada trata de evitar. Si ya programas en BASIC, seguramente ya poseerás hábitos creados con la instrucción GOTO. Si es así te aconsejo que intentes "desintoxicarte" de ella.

La que sigue es la declaración de tipos. Para empezar tendremos que ver lo que es un tipo. Un tipo de datos indica de que tipo es un dato (valga la redundancia). Si, por ejemplo, nuestro programa va ha hacer una suma, lo normal sería usar datos de tipo numérico. TPascal tiene varios tipos de datos, como números enteros, reales, tipo carácter, cadena de caracteres, booleanos, etc... Ya los veremos más adelante según los necesitemos. Pero la función de éste bloque declarativo, no es precisamente el de declarar estos tipos de datos que he citado, ya que éstos ya están declarados. En éste bloque podremos declarar nuestros propios tipos de datos, ya veremos algunos ejemplos más adelante.

Ahora, le toca el turno a la declaración de constantes. Una constante es un dato que no cambia durante la ejecución de un programa. Imagina, por ejemplo, que vamos a hacer un programa que calcule el area encerrada en una circunferencia. Para ello necesitamos el valor del número PI (3.1416). Evidentemente PI vale lo mismo siempre durante todo el programa. de forma que declarariamos PI como una constante. (TPascal ya contiene la constante PI declarada de antemano). El inicio de la declaración de constantes se indica mediante la palabra reservada CONST:

CONST PI = 3.1416;

Nos enfrentamos ahora al bloque de declaraciones quizás más importante de todos; El de declaración de variables. Una variable es una zona de la memoria en la guardamos un dato. Puedes imaginar la memoria como los buzones de un bloque de casas. Cada buzón es representado por el número de piso que le corresponde, por ejemplo, el primer buzón se llamaría 1ºA. Dentro de cada buzón podemos meter información. Bien, una variable sería similar a uno de estos buzones, sin embargo estos buzones no se llaman 1ºA ni 3ºB. Nosotros asignamos el nombre que queremos darle. Pero veamos su utilidad en la práctica. Imagina que vamos ha hacer un programa para calcular el area de un triángulo. Esto es lo que debería hacer el programa:
** Indicar la longitud de la base - Indicar la longitud de la altura - Realizar la operación BASE*ALTURA/2 - Mostrar los resultados**
Pues bien, cuando ejecutamos el programa, no podemos saber que datos introducirá el que lo utilize, por lo que tendremos que usar variables para poder almacenarlos temporalmente. En nuestro ejemplo crearíamos una variable llamada BASE, otra llamada ALTURA, y otra tercera para almacenar el resultado que llamaremos RESULTADO. De esta forma introducimos la longitud de la base en la variable BASE, y la de la altura en la variable ALTURA. Tras ésto, almacenamos al resultado de la operación en la variable RESULTADO. Lo mejor si no lo ves muy claro, es que sigas leyendo y veas luego el ejemplo. El inicio del bloque de declaración de variables se indica con la palabra reservada VAR.

VAR altura : integer;

El tipo INTEGER indica que se trata de un número entero, pero veremos ésto un poco más adelante.

Los dos bloques de declaración siguientes, son para declarar funciones y procedimientos. No entraremos muy en profundidad en este bloque todavía, pero trataré de dar una visión general. Una función o procedimiento es un "trozo" de código que escribimos a parte y que llamamos desde el programa principal. Esto se hace para evitar escribir trozos de código repetido dentro del programa principal. Imagina que estas programando un menú de opciones para un programa. Supón que cada vez que pulsas una opción, tienes que borrar la pantalla y cambiar el color de la misma. Sin funciones/procedimientos lo harías así:

PROGRAMA - Presenta menú en pantalla - espera a que el usuario seleccione una opción - si ha seleccionado la primera - borra la pantalla - cambia de color la pantalla - haz lo que pedía la opción - si ha seleccionado la segunda - borra la pantalla - cambia el color de la pantalla - haz lo que pedía la opción
- resto de programa... ...

Las instrucciones que están más hacia la derecha indican que se ejecutarán sólo si se cumple la condición. Esto se llama identación y es una práctica muy aconsejable. Usando funciones/procedimientos podríamos hacerlo de esta otra manera:

PROCEDIMIENTO A: borra pantalla cambia color

PROGRAMA - Presenta menú en pantalla - espera a que el usuario seleccione una opción - si ha seleccionado la primera - llama al procedimiento A - haz lo que pedía la opción - si ha seleccionado la segunda - llama al procedimiento A - haz lo que pedía la opción
- resto de programa... ...

Veremos esto con más en detalle cuando lo necesitemos.

.- BLOQUE DE INSTRUCCIONES. Este es, por fin, el bloque que contiene el programa en sí. Este bloque comienza siempre con la palabra reservada BEGIN y termina con END., no olvidando nunca el punto que le sigue. TPascal posee una serie de palabras reservadas que permiten controlar el flujo del programa. Además, existen una serie de funciones y procedimientos que TPascal lleva consigo que nos permitirán hacer un montón de cosas, de hecho la instrucción WRITELN que vimos antes, no es más que un procedimiento. Pero, ¿qué es un programa? Bien, parece que la misión principal de un programa debería ser la de procesar datos, pero, ¿dónde ponerlos mientras los usamos? indudablemente en variables. El proceso de guardar un dato en una variable se llama asignación, y se lleva a cabo mediante el signo de operación := (dos puntos y el signo igual). veamos un ejemplo:

BASE := 10; ALTURA := 5;

y por supuesto, podemos almacenar en una variable el resultado de una operación aritmética:

RESULTADO := BASE * ALTURA;
CUADRADO := 2 * 2;

hay numerosas operaciones disponibles, pero ahora sólo usaremos las básicas para el manejo de números enteros, que son las que están en la figura 3. En las operaciones aritméticas, es posible la inclusión de paréntesis, por ejemplo:

RESULTADO := (BASE * ALTURA) DIV 2;

NUESTRO PRIMER PROGRAMA UTIL

Teclea el programa de la figura 4 y ejecutalo. Verás como el ordenador es perfectamente capaz de resolver este pequeño problema de cálculo de áreas. Realmente el ordenador es capaz de resolver problemas mucho más complejos de lo que puedas imaginar, sin embargo eres tú el que debe decirle todo lo que debe hacer paso a paso, hasta el más mínimo detalle. TPascal nunca hará nada que nosotros no le hallamos dicho que haga, así que nunca supongas que TPascal hará algo porque es muy evidente que debiera hacerlo. Pasemos ahora a comentar el programa de cálculo de áreas. La primera línea es la cabecera y ya la conocemos, así que pasaremos a la siguiente línea. Nos encontramos con la palabra reservada USES, que como recordarás era para declarar unidades. Realmente lo que hacemos con esta línea es decirle TPascal que vamos a usar la unidad CRT. Esta unidad es la encargada de la gestión del monitor en modo texto. Después nos encontramos con la declaración de variables, es decir le decimos al TPascal que variables vamos a usar en el programa. TODAS las variables que vamos a usar en nuestro programa deben ser declaradas aquí y definiendo su tipo. En nuestro caso hemos definido tres variables de tipo entero (sin decimales). Después del BEGIN, en el bloque del programa, la primera instrucción que encontramos es CLRSCR. Este procedimiento está definido en la unidad CRT (es por ello por lo que la declaramos antes) y su misión es borrar la pantalla. Encontramos en esta línea algo que no habíamos visto hasta ahora, se trata de un comentario. Un comentario es una frase o frases que se colocan entre los signos (* y *) o entre llaves { }. a lo que se incluya dentro de los signos de comentarios, TPascal no le hace ningun caso, de forma que podemos comentar cada línea o grupo de líneas para que nuestro programa gane en claridad. Veamos algún ejemplo de comentario:

(* Esto es un comentario *) { esto también es un comentario } { writeln ('Esta línea no se ejecutara porque es un comentario') }

Seguidamente vemos que hay dos asignaciones de variables, en ellas introducimos el valor de la base y de la altura, de forma que en la línea que sigue, calculamos el area del triangulo basandonos en los valores almacenados en las variables BASE y ALTURA. La siguiente línea merece un comentario algo más detallado, ya que este WRITELN tiene alguna diferencia con el que vimos antes. Con la instrucción WRITELN no sólo podemos imprimir texto en pantalla, sino que también podemos imprimir valores de variables. Para diferenciar lo que es un texto de lo que es una variable, encerremos los textos entre comillas, y las variables se dejan fuera de las comilla. Notese que el texto a imprimir entre comillas está separado de la variable mediante una coma. Veamos algunos ejemplos con WRITELN y lo que imprimiría.

WRITELN(resultado); Imprime valor de la variable RESULTADO.
WRITELN('resultado'); Imprime la palabra RESULTADO.
WRITELN('el resultado es ',resultado,' metros'); Imprime EL RESULTADO ES X METROS, donde X es el valor de la variable RESULTADO.

Damos fin aquí a esta primera entrega del curso de Turbo Pascal, en el siguiente número nos adentraremos en otro tipo de números, los reales, y veremos la forma en que podemos interactuar con el usuario. Hasta entonces, practica con el TPascal y diviertete.

LA TARJETA VGA

EL BIOS DE VIDEO

La palabra BIOS significa Basic Input Output System o sistema básico de entrada y salida. Para todo aquel que no domine la lengua anglosajona, tan compleja abreviatura puede no significar nada, especialmente lo de la entrada y la salida. Entrada y salida...¿de qué?. Incluso cuando uno entiende perfectamente que es el BIOS y que función tiene dentro del ordenador, mira atrás a su definición y vuelve a cuestionarse su significado.

En realidad el BIOS se encarga de ser un interface con nuestros programas. Me explico, para no caer en malentendidos. Cuando una pieza de hardware se conecta a nuestro ordenador, en una de las ranuras de expansión, éste queda integrado en nuestra máquina gracias al BUS. Este último es la línea de datos que une todos los componentes del hardware con el microprocesador para recibir por parte de éste los datos y las señales de control tan necesarias para que todo el ordenador actúe de forma conjuntada y precisa.

Pues bien, los componentes hardware que se pueden contectar al ordenador son tan variados y cambian y mejoran con tanta rapidez que es imposible "entrenar" previamente al procesador para que sepa interaccionar con todos ellos. La solución pasa por hacer que cada componente lleve su propio programa controlador, de forma que cuando queramos manejarlo desde nuestros programas lo hagamos mediante esta porción de software que viene incluido ya en el propio elemento hardware. Además, quien mejor que el propio fabricante para hacer ese interface, al que también se le llama firmware -del inglés firme o innamovible- ya que el quien mejor conoce como funciona ese periférico.

Este firmware es conocido siempre como BIOSs y por supuesto se guarda como memoria ROM, o de solo lectura, que como ya todos sabemos es inborrable. De esta forma tenemos el BIOS de la tarjeta de vídeo, el de la controladora de discos, el de la tarjeta de audio, el de la tarjeta de red, el del scanner, etc., los cuales contienen rutinas o trozos de programa que están especializados en interraccionar con las distintas peculiaridades de el hardware del componente en cuestión.

En el caso que nos interesa en esta sección, la tarjeta de vídeo, el BIOS es conocido como VBIOS, significando "vídeo" la V que se le añade delante. La VBIOS contiene las diferentes rutinas que se encargan de darnos los servicios necesarios a la hora de manejar desde nuestros programas las salidas por pantalla de información gráfica o alfanumérica.

De esta manera, tan sólo tendremos que indicarle al VBIOS que haga lo que nosotros queremos y el lo realizará para nosotros y sin posibilidad de incompatibilidades, como las que ocurren cuando nosotros, los programadores, nos saltamos el BIOS y programamos la tarjeta por nuestra cuenta y a nuestro gusto.

Pero esto tiene un inconveniente. Si usamos el BIOS en nuestra programación, deberemos ser conscientes de dos cosas. Primero, que el número de servicios que éste es capaz de darnos es limitado. Segundo, que el cumplimiento de estas tareas es siempre más lento que si el hardware se programa de una forma directa, aunque existan riesgos. Riesgos que, por otro lado, no son tan definitivos como para impedirnos que hagamos este tipo de programación, tan sólo deberemos ser cuidadosos con lo que hacemos y siempre siguiendo unas reglas preestablecidas de compatibilidad. Quede claro, pues, que el BIOS es el intermediario o interface entre los elementos hardware y el usuario o programador. Podemos utilizarlo cuando nos interese, por ejemplo en los servicios en los que cumpla con rapidez y eficacia en nuestros programas, o saltárnoslo cuando queramos más velocidad o simplemente cuando queramos hacer algo especial, fuera de lo estándar. En cuanto al nombre, debemos conformarnos con él aunque en realidad no entendamos muy bien que quiere decir eso de sistema básico de entrada y salida. Yo lo hubiese llamado algo como "conjunto de rutinas interface" para indicar ese nexo de unión entre el programador y el hardware.

COMO DE USA EL BIOS

Como hemos visto, el BIOS está siempre ahí, dispuesto a servir nuestras necesidades siempre que lo necesitemos, pero ¿cómo utilizo los servicios que éste me ofrece?. Los que estén siguiendo el apartado "el MSDOS y el programador de videojuegos" ya tendrán una idea de como se puede llegar a con-

seguir esto mediante el uso de las interrupciones, pero para los que no estén leyendo esta serie de artículos voy hacer aquí una pequeña introducción.

Todos los lenguajes de programación, entre los que se incluyen el ensamblador, el C y el BASIC por ejemplo, tienen posibilidades de llamar a una parte de programa y después volver de él continuando en el mismo punto desde el que saltamos, cosa que se conoce como "llamada a una subrutina". Para ello tenemos ordenes como CALL y GOSUB por ejemplo.

Estas instrucciones provocan un salto a una dirección de programa, la ejecución de una sección de código y una vuelta al programa principal cuando se encuentra con una instrucción del tipo RET o RETURN (volver). Dichas secciones de programa se conocen como subrutinas y casi siempre realizan una función repetitiva. Para no tener que copiarlas innumerables veces en el programa principal se ponen a parte y se las llama cada vez que nos hagan falta.

Un ejemplo: Realizamos un programa que muestra en pantalla las edades de los alumnos de una clase. En la clase hay 120 alumnos por lo que cada vez que la pantalla se llena de información, tenemos que borrarla. Nos damos cuenta, a la hora de crear el programa, que necesitamos borrar la pantalla 13 veces durante el transcurso del funcionamiento del mismo. Eso hace que tengamos que escribir ese número de veces la porción de programa que se encarga de la limpieza de la pantalla. Un latazo, ¿verdad?. Gracias a que somos inteligentes, hacemos la subrutina de borrado de pantalla una sola vez y la llamamos desde el programa principal tantas veces como sea necesario. (ver figura)

El BIOS hemos dicho que era un conjunto de servicios disponibles para nuestros programas y para el sistema operativo de nuestro ordenador, que siempre está disponible para su uso, pues bien, el BIOS se constituye de un gran conjunto de subrutinas a las que se puede llamar y que realizan funciones como borrar pantalla, escribir caracteres, dar color, dibujar pixel, hacer scroll de pantalla, cambiar de página de vídeo, manejar el cursor, etc.



Gracias al BIOS el PC puede manejar las tarjetas de vídeo

En el mundo del PC, sin embargo, al igual que en algunos otros ordenadores, existen dos tipos de subrutinas: Las propiamente llamadas subrutinas y las interrupciones. Ambas tienen el mismo funcionamiento (llamada-retorno) pero son diferente en la forma en que son llamadas y en la forma en que se retorna de ellas. De todas formas, en el curso del lenguaje ensamblador de irán desglosando ambas. Por ahora nuestra intención es que te quede claro el concepto de subrutina y su utilidad.

LAS INTERRUPCIONES

Mediante el uso de este sistema de llamada a una subrutina o servicio del BIOS, en este caso el VBIOS como hemos visto anteriormente, accedemos a todas las funciones que éste puede darnos para ser usadas desde nuestros programas. El VBIOS es pues un gran conjunto de subrutinas a las que les vamos a asignar un número de orden para conocerlas, diferenciarlas y poder hacer uso de ellas. Tendremos pues la función 0, 1, 2, 3, 4, etc. Pero en lugar de usar la notación decimal vamos a hacer uso de la hexadecimal que es la preferida de todos los programadores. Así tendremos la 00h, 01h, 02h, etc. Veamos como llamar a una interrupción BIOS desde nuestro propio código. Al igual que el MSDOS, el VBIOS tiene un vector de interrupción general que se conoce como 10h. A través de él tenemos la posibilidad de usar las funciones de las que hablábamos antes de la siguiente forma:

```
MOV AH,número servicio
INT 10H
```

En el registro AH se carga el valor de la función a la que queremos llamar en concreto dentro de la interrupción 10h que reune los servicios de vídeo del PC.

En el próximo número de Código Juego comenzaremos a ver los diferentes servicios con ejemplos de su utilización para nuestros juegos y como podemos sacar partido de ellos en nuestras aplicaciones.

Hasta Septiembre, amigos.

Curso de programación bajo Windows

Profesor

Leopoldo Fernández

Windows, de la compañía norteamericana Microsoft, se ha convertido en los últimos años en un estàndar de los entornos de trabajo para el PC.
Desde un principio pretendió romper con los sistemas operativos tradicionales y su pobre interface con el usuario, mejorando y agilizando la manera de trabajar con un ordenador mediante el uso exaustivo de gráficos y explotando toda la potencia de un periférico que resulta de gran utilidad como es el ratón.
Aunque la mayoría de las aplicaciones creadas para él son de gestión, ya empieza a haber buenos juegos.

MICROSOFT WINDOWS 3X

Windows nace en el seno de Microsoft en el año 1985, aunque sus orígenes se remontan a las investigaciones realizadas en los años 70 en el Xerox Reseach Center de Palo Alto (PARC), California. Una de sus virtudes esenciales, que le ha acompañado desde su primer diseño, consiste en hacer que la pantalla del ordenador funcione en modo grafico, en vez de hacerlo en modo texto, como en MS-DOS.

Como su nombre indica el modo gráfico permite representar los datos, programas, herramientas, procedimientos, etc.., de forma gráfica y, en consecuencia, de una manera más asequible y rápida de entender.

Otra característica esencial es su forma peculiar de organizar la información interna que está a disposición del usuario: Windows dispone de un sistema jerárquico de menús (listados de opciones por pantalla), con lo que no es necesario teclear las habituales series de comandos y órdenes del MS-DOS, con su sintaxis rígida y difícil de memorizar, como conocen todos los usuarios de tan distinguido sistema operativo.

Así pues, Windows tiene en cada momento un sistema de menús que nos representan en pantalla todas las operaciones que en ese momento están disponibles. Además, estos menús funcionan siempre de una forma similar, lo que nos permite emplear fácilmente distintas aplicaciones, como hojas de cálculo, tratamientos de texto, etc..dedicando un tiempo mínimo al preceptivo aprendizaje preliminar. Windows unifica la forma de operar.

Por otra parte y para facilitar la "reconversión" de los usuarios avezados en el manejo tradicional de las aplicaciones informáticas, la selección de los elementos gráficos o los menús se puede hacer desde el teclado... aunque es mejor utilizar el ratón, ese pequeño dispositivo que está siempre preparado para apuntar sobre zonas de pantalla.

El entorno gráfico de Windows se sirve de zonas rectangulares en la pantalla, es decir ventanas, y de aquí su nombre. Estas ventanas contienen las aplicaciones o los documentos que se encuentran en los directorios. Las ventanas ocupan toda o una parte de la pantalla, según se necesiten y pueden superponerse, solaparse, ubicarse en zonas adyacentes de la pantalla etc...

Windows se acerca a nuestro modo de gestionar la información y es capaz de adaptarse a fórmulas convencionales. Es como so estuviera dispuesto a aprender nuestro estilo de trabajo habitual.

En lugar de tener que aprender a trabajar como el ordenador, Windows enseña al ordenador a trabajar como nosotros. Crea un pantalla que es equivalente a nuestra propia mesa de trabajo. En ella están situadas las carpetas, los documentos y los útiles de escritorio que utilizaremos para nuestro que hacer.

MS-DOS es un sistema mono-tarea, quiere decir esto que sólo puede hacer una cosa a la vez, mientras que Windows puede estar realizando varias acciones simultáneamente, y esto es necesario pir que nosotros también somos multitarea. Lo somos en dos niveles. El primero podemos llamarlo multitarea pura: somos capaces de conducir y hablar a la vez, caminar y masticar chicle, comer y ver la televisión...

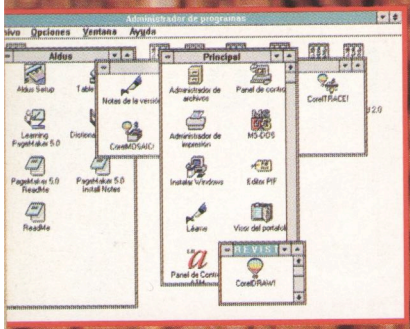
Semejante capacidad en nuestro ordenador nos ahorra tiempo y mejora nuestra productividad.

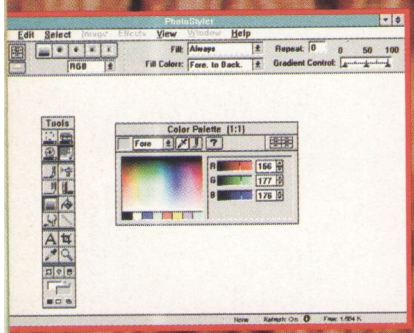
El segundo nivel de multitarea se refiere a los cambios rápidos de actividad. Estamos trabajando sobre un documento y suena el teléfono. Inmediatamente lo cogemos y contestamos a la llamada, para volver a continuación al punto en el que estábamos. Windows nos permite esto. Por el contrario MS-DOS exige guardar el documento en su carpeta, ésta en el archivador, cerrarlo y atender a la llamada, pare iniciar después el proceso contrario...

Uno de los problemas que ha surgido con el MS-DOS es la disparidad de formas que cada programador ha ideado para acciones similares. Pasar de un procesador de textos a otro puede ser una experiencia traumática, porque con la misma combinación de teclas que uno nos permite salvar en disco un documento, el otro nos muestra una pantalla de ayuda.

Windows viene a poner orden en este caos.

Todas las aplicaciones de windows tienen una manera uniforma y coherente de actuación, en sus funciones básicas. Hay toda una serie de controles que responden en cualquier programa de una manera previsible, esto no sólo es cómodo y evita frustraciones, tiene una traducción inmediata en costes de formación y repercute en la productividad.





He comentado que Windows permite realizar varias tareas a la vez y ejecuta varios programas simultáneamente. Esto puede ser muy importante, pero Windows va un paso más allá: es capaz de integrar esas aplicaciones.

A medida que han ido apareciendo programas para las actividades más variadas los usuarios de PCs han empezado a usar más programas. Con mucha frecuencia necesitamos integrar datos de unos programas en otros y aquí es donde empiezan las dificultades, cada programa tiene su forma propia de guardar los documentos, que no es compatible con las de otros programas.

Tradicionalmente de han propuesto dos soluciones a este problema, Una consiste en diseñar varios programas integrados en uno sólo, ésta puede ser una solución adecuada, pero a medida que nuestras necesidades de informatización crezcan, se nos empezará a quedar corto uno de sus componentes.

La otra solución es hacer programas que realicen bien una sola tarea, pero que sean capaces de utilizar documentos de otros.

Windows tiende a ser la tercera solución, enlazada distintos programas y documentos que estén en uso. De este modo compramos el procesador de texto, la hoja de cálculo o la base de datos que más nos gusten o que consideremos más adecuados, sin reparar

o no que sean del mismo fabricante. Windows va a permitir que se pasen información entre ellos como si se tratase de un mismo programa integrado, pero con la diferencia que para cada componente habremos optado por lo mejor. Al menos según nuestro criterio.

LA PANTALLA EN WINDOWS.

Al arrancar Windows, aparece en pantalla lo que se denomina el Administrador de Programas, que es el centro de todas las sesiones de trabajo con el ordenador.

Merece la pena detenerse en la propia pantalla del ordenador y analizar algunos componentes que Windows incorpora y que serán comunes a todos los programas que utilicemos bajo este entorno gráfico.

En primer lugar conviene destacar que toda la superficie de la pantalla conforma el llamado "escritorio", concebido a semejanza de nuestra propia mesa de trabajo. La superficie de este escritorio está ocupado por ventanas y por iconos que corresponden, respectivamente a las aplicaciones o documentos y a su representación minimizada.

Entendiendo por Aplicación: Un programa de ordenador diseñado para ayudar al usuario a realizar un determinado tipo de trabajo. Una aplicación, traba con texto, números, gráficos o una combinación de todos ellos.

Icono: Un símbolo o pequeña representación gráfica, que en Windows representa una aplicación, un documento o una función que se pueda realizar en ese momento.

Minimizar: Hacer que un programa en ejecución bajo Windows, en una ventana o en pantalla completa, pase a estar como Icono en la parte inferior

de la pantalla.

Nos encontramos con dos tipos de ventanas: Las especializadas en aplicaciones y las que contienen documentos. Las primeras corresponden a área de trabajo de una aplicación, por ejemplo un procesador de texto, mientras que las segundas corresponden a distintos documentos producidos por esa u otra aplicación.

APLICACIONES DE SISTEMA INCLUIDAS EN WINDOWS.

_Administrador de Programas: Gestiona y manipula los Iconos permitiendo organizar y arrancar aplicaciones.

_Administrador de Ficheros: Gestiona la tarea relativas a ficheros y directorios, proporcionando mapas de los discos locales y de red, y de los directorios. Las operaciones de copia y movimiento de ficheros se efectúan con el ratón, la ejecución de aplicaciones

y la impresión de ficheros, también se controlan directamente con movimientos del ratón.

_Administrador de Impresión: Visualiza las colas de impresión, permite modificar el status de las tareas pendientes, arrastrar y soltar documentos desde una cola de impresión a otra.

_Panel de Control. Gestiona la conexión de impresoras en red, parámetros de los puertos de comunicaciones, fecha y hora del sistema, instalación de fuentes, personalización del interfaz, etc.

_Portapapeles: La acción de copiar para después pegar es una de las que

se ejecuta con más frecuencia en el entorno Windows: copiar datos de una celda a otra en una hoja de cálculo, copiar datos repetitivos en un mismo documento de procesador de texto, crear un logotipo en un programar gráfico y pegarlo dentro de un documento de texto... todas esas opciones se realizan dentro del portapapeles.

El portapapeles es una utilidad dentro del entorno Windows que permite reservar una porción de memoria del sistema donde reside transitoriamente la información que se copia antes de ser pegada. La naturaleza de los datos pegados es heterogénea, desde texto a dibujos, números, imágenes escaneadas, etc. El portapapeles es el encargado de almacenarla transitoriamente.

Ver el contenido del portapapeles es posible haciendo doble click con el ratón en el visor del portapapeles que incluye el grupo principal del Administrador de Archivos; y también algunos programas incluyen esta posibilidad en forma de comando. En su interior estará la información copiada previamente en cualquier otra aplicación, un texto, un dibujo, una imagen, números...etc.

El contenido del portapapeles se puede pegar en cualquier aplicación que se desee(si la aplicación destino gestiona la información que se va ha pegar. Por ejemplo, el block de notas no gestiona dibujos, y no admite que se le pegue uno) y tantas veces como se quiera mientras la información esté en el portapapeles, de modo que un dato copiado en una hoja de cálculo, puede ser pegado en un procesador de texto, en el block de notas, en otra hoja distinta, etc. sin necesitar en ningún caso de

cerrar una aplicación para entrar en otra, pues pueden estar abiertas las dos a la vez.

APLICACIONES DE USUARIO INCLUIDAS EN WINDOWS.

_Microsoft Windows write: Procesador de textos ejecutivo.
_Microsoft Windows Paintbrush: Programa artístico a todo color.
_Terminal: Programa de comunicación/emulación de terminales asíncronos que incluye transferencia binaria de ficheros y posibilidad de asociar macros hasta a 32 teclas.
_Agenda: Calendario mensual y diario con anotación de citas y alarma acústica para recordarlas en su momento.
_Calculadora: Convencional y científica, que permite efectuar cálculos en binario, octal, hexa y decimal.
_Grabadora: de macros sencilla, que recoge los movimientos de ratón y pulsaciones de teclas para generarlas automáticamente siempre que se desee. Permite automatizar tareas en aplicaciones basadas en ventanas.
_Reloj: analógico y digital. Puede visualizarse en cualquier lugar de la pantalla.
_Fichero: Programa que gestiona un fichero tarjetero indexado. Incluye marcación telefónica automática. Las fichas (tarjetas) pueden contener texto y /o gráficos.
_Block de notas: Editor de texto adecuado, para la escritura rápida de notas, mimos y otros trabajos de poca extensión, incluye el estampado automático de fecha y hora en el documento.

MEJORAS DE WINDOWS 3.1 FRENTE A SU PREDECESOR 3.0

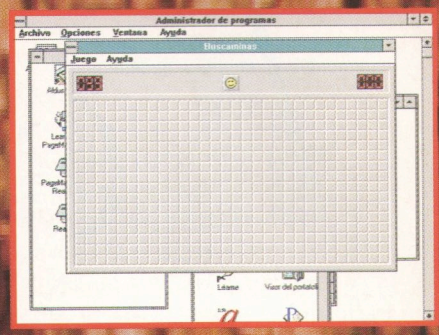
El 6 de abril de 1992, tras una inmensa prueba de la versión "beta" por parte de miles de usuarios, Microsoft presento la versión 3.1 de su entorno gráfico. A primera vista, ha cambiado el logo de windows, ocupa un disco más, la documentación es deferente...pero ¿Cuáles son las verdaderas diferencias entre la versión 3.1 y su predecesora?
Durante un año y medio, Windows 3.0 ha conseguido justa fama por méritos propios. Todo el mundo del PC se ha volcado en el entorno gráfico de Microsoft, pero tampoco la versión 3.0 es perfecta y prácticamente todos hemos tenido que sufrir los "Errores Irrecuperables de la aplicación", su lentitud, su mala resolución al imprimir (a no ser que se recurriese a un gestor de tipos de letra (fuentes) como el Adobe Type Manager), etc.

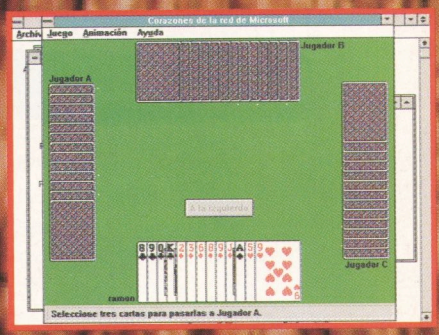
O. L. E. ¡olé!

El OLE (Object Linking and Embedding, enlace e incursión de objetos) es una tecnología que permite hacer maravillas con los documentos procedentes de otras aplicaciones sobre un documento "padre". Veamos un ejemplo: supongamos que se escribe un documento con un tratamiento de textos (que tenga capacidad OLE, por supuesto) y se desea incluir una ilustración. Se debe seleccionar insertar objeto en el menú de edición. Se selecciona el tipo de p
objeto que se desea insertar (en éste caso un dibujo del Paintbrush). Se busca en el disco duro el fichero que contiene el dibujo y se selecciona. En este punto, se puede "enlazar" el objeto (establecer una conexión entre el fichero en disco y el documento que se está procesando, en cuyo caso las modificaciones posteriores en el fichero del dibujo se reflejan automáticamente en el documento) o "incluirlo" (en ese caso, el dibujo formará parte del documento y no tendrá nada que ver con el fuchero de donde se tomó).
En este caso se selecciona "enlace" por que el dibujo no es definitivo y necesita unos retoques. Después de este sencillo proceso se tiene la ilustración en el documento y para modificarla sólo ha de hacerse un doble click sobre el objeto OLE. El programa de tratamiento de texto llamará al programa que creó el objeto para permitir su modificación.
Un ejemplo de la potencia del OLE, nos lo brinda la versión 2.0 de Microsoft Word para Windows. Soporta OLE y gestiona las ilustraciones por este sistema; no sólo ilustraciones, sino que también gráficos de tipo draw, chart, word-art u otro tipo de datos no imprimibles, como los de sonido. Un documento puede contener voz y música...

LO QUE FALTA.

A pesar de las muchas mejoras (las comentadas y las que se quedan en el tintero por razones de espacio o por ser menos importantes), hay aún detalles que esperan otro tipo de solución.
Un ejemplo claro de esto último es el Administrador de Programas. El administrador de programas necesita algún retoque como permitir incluir grupos dentro de cada grupo (al estilo del entorno New Wave, de la compañía Hewlett-Packard).
También resulta molesto que siga sin permitir cambiar el tamaño de la paleta por defecto. Si se tiene una tarjeta gráfica con un coprocesador gráfico y permite utilizar 256 coloers en todas las resoluciones, resulta irritante que el entorno de trabajo utilice siempre una paleta de veinte tinteros. Como resultado Paintbrish, sigue estropeando cada imagen de 256 colores que utilizan la paleta por defecto.
Si bien microsoft o alguien decide corrogir este problema, ya puestos, que permitan modificar los colores de la padeta. De este modo los usuarios de las tarjetas VGA normalitas podran trabajar con 16 colores y no con 8 y otros 8 casi iguales.
Aunque Microsoft ha incluido funciones de descompresión de datos, según los algoritmos Lempel-Ziv, seria aconsejable que incluyera también las de compresión, y que las utiluzara internamente para retener más datos en memoria.
Un procesador rápido sería capaz de comprimir/descomprimir bitmaps en tiempo real y de esta forma almacenar fácilmente imágenes unas 5 veces mayores, o textos del doble de tamaño, o grandes tablas de datos.
También el cache de disco podria aprovechar estas téchicas de compresión/descompresión para retener más datos en el cache.





Una de las características más destacables de Windows es el OLE, que significa Object Linking And Embedding, que en castellano viene a ser algo así como Incrustación y Vinculación de Objetos.
Esta Característica permite la posibilidad de transportar objetos entre las diferentes aplicaciones a través del portapapeles con el consiguiente ahorro que esto supone para el paso de información entre aplicaciones.

Lenguaje C

Profesor

José A. Brihuega

OPERADORES Y CONTROL DE FLUJO

Las instrucciones de control son parte fundamental de cualquier lenguaje de programación. C es un lenguaje secuencial. Esto quiere decir que las instrucciones se ejecutan una tras otra en el orden en que están escritas. Las instrucciones de control cambian esa forma de actuar, reconduciendo el flujo del programa como sea más conveniente. Sin ellas, la capacidad del lenguaje quedaría reducida a la mínima expresión. Las instrucciones de control están estrechamente vinculadas a las expresiones con operadores lógicos, es decir, aquellas operaciones cuyo resultado no es numérico, sino del tipo Verdadero - Falso. Normalmente, el flujo del programa se reconducirá basándose en el resultado verdadero o falso se este tipo de operadores.

COMPARACIONES

Las condiciones más sencillas son las que se hacen con los operadores de comparación. Estos operadores comparan dos valores. Dichos operadores son los siguientes:

<menor que >mayor que
<=menor o igual
>=mayor o igual ==igual
!=distinto

La expresión

a > b

se evalúa a verdadero si el valor de la variable a es mayor que el de b. Como siempre en C, está permitido introducir espacios, incluso saltos de línea, entre los operadores y sus operandos, para hacerlas más legibles. Conviene hacerlo, pero no es obligatorio. Por supuesto, se puede mezclar variables y constantes sin problemas:

a == 3
10 > dato
valor >= 0

Hay que tener cuidado con el operador de igualdad (==), pues se puede confundir con el de asignación (=). Se puede también comparar expresiones aritméticas y, en general, cualquier cosa que produzca un resultado numérico (incluidas las funciones).

dato > b + 10
espacio() > 10000

Los operadores aritméticos se evalúan antes que los de comparación (se dice que tienen mayor prioridad). Es por ello que no hace falta encerrar la expresión b + 10 entre paréntesis para que la comparación del ejemplo anterior se ejecute correctamente. Primero se sumará 10 al valor de la variable b, y posteriormente, se comparará el resultado con la variable dato. Estrictamente, los operadores de comparación sólo trabajan con expresiones numéricas (int, unsigned, float, etc.). Sin embargo, ya se ha hablado aquí de que el tratamiento de los caracteres en C es totalmente numérico, es decir, el lenguaje trata a los caracteres por su código ASCII, que es un número entero de 0 a 255. Por tanto, no hay inconveniente en usar los operadores de comparación con los caracteres (tipo char). De hecho, lo que se comparan son los códigos ASCII. La expresión siguiente:

inicial == 'A'

es totalmente correcta. Rizando el rizo, se puede incluso mezclar expresiones char con expresiones numéricas enteras (no float ni double). La siguiente expresión es equivalente a la anterior:

inicial == 65

Es válida aunque inicial se haya declarado como char. Es importante el uso de las comillas simples en las constantes de tipo carácter. No se debe nunca usar para este propósito las comillas dobles (").

EXPRESIONES LOGICAS

Se puede combinar varias comparaciones en una expresión lógica del tipo "si llueve Y salgo a la calle...", "si llueve O me ducho...". Obsérvese que en ambos casos se trata de combinar dos condiciones en una sola. - El operador Y da resultado verdadero si ambas condiciones son verdaderas al mismo tiempo. En C se representa por el operador &&.

a > 0 && b > 0

Esta expresión es verdadera sólo si ambas variables son mayores que cero. En este otro caso

dato == 0 && x != 0

es verdadera si la variable dato es cero y, al mismo tiempo, x es distinta de cero. El operador && no se debe confundir con &, que es otro operador válido, con un uso totalmente dis-

tinto. - Al operador O le basta con que una de las condiciones sea verdadera. Se representa con dos barras verticales (||).

a > 0 || b > 0

Esta expresión lógica da verdadero si cualquiera de las variables a o b es mayor que cero, o bien ambas al mismo tiempo. Al igual que el anterior, este operador || no se debe confundir con |, que tiene otra función distinta. - Existe otro operador lógico: la negación, representada por un signo de admiración (!), que invierte el resultado de una condición: lo verdadero se hace falso, y lo falso pasa a ser verdadero. La expresión

!(dato > 0)

dará resultado verdadero cuando la variable dato no sea mayor que cero. Obsérvese la utilización de los paréntesis. La negación tiene mayor prioridad que la comparación, por lo que tenemos que usar los paréntesis si queremos que se haga antes la compara ción. Si no se ponen paréntesis, el ! sólo afectaría al operando directamente a su derecha (en el ejemplo, sólo dato), lo que daría un resultado incorrecto. Las expresiones lógicas pueden ser tan complejas como se necesite. En el cuadro adjunto se presentan algunas expresiones válidas.

(PONER AQUI CUADRO1)

Son interesantes en estos ejemplos: el uso de valores devueltos por funciones, la mezcla de elementos numéricos y de carácter, el uso de los paréntesis para forzar el orden de las operaciones, y por último, la combinación de los diferentes operadores (como en el último ejemplo). El uso de espacios y saltos de línea entre los elementos de la expresión es totalmente libre. No se usa el punto y coma para terminarlas, pues no son instrucciones completas del lenguaje, sino parte de ellas.

LA INSTRUCCION IF

La instrucción de control de flujo más básica es if, cuya forma general es:

if (<expresión>) <instruccion1> [else <instruccion2>]

* Nota: Cada vez que representemos la sintaxis de una instrucción, usaremos los ángulos (<>) para representar el lugar donde se deben poner los operandos de la instruc ción. Asimismo, los corchetes ([ ]) señalarán las partes que se pueden omitir. Es importante recordar que ni los ángulos, ni los corchetes, ni las palabras explicativas (que aparecerán entre los ángulos) pertenecen realmente a la instrucción; aparecen con intención simplemente explicativa. La instrucción if evalúa la <expresión> (normalmente se trata de una expresión lógica) y actúa en consecuencia: si la expresión es verdadera, se ejecuta la <instruc ción1>; si es falsa, se ejecutará la <instruc ción2> (la que va tras la cláusula else). En el caso que no haya else (es opcional) y la expresión resulte falsa, no se lleva a cabo acción alguna, pasando el control a la siguiente instrucción del programa. Sin embargo, la <instrucción1> no se puede omitir. Los paréntesis que delimitan la expresión deben ponerse siempre. Por ejemplo, la siguiente instrucción

if (x == 0) y = 1;

asigna a la variable "y" el valor 1, siempre que x valga cero. En caso contrario, no hace nada, continuando normalmente con la siguiente instrucción del programa. Obsérvese la diferencia entre el operador de comparación (==) y el de asignación (=), y la obligación de terminar las instrucciones (tanto la del if como, si existe, la del else) con punto y coma. Como siempre, ambas instrucciones se pueden sustituir, si se necesita, por bloques de instrucciones entre llaves ({}).

if (x != 0) { y=raiz_cuadrada(x); x = 1; } else x = -1;

Si x no es cero, se ejecuta la función raiz_cuadrada() y se le da valor uno. En caso contrario, se le da valor -1. Aunque la forma de escribir las instrucciones en C es totalmente libre, es costumbre, para facilitar la lectura, escribir los bloques como se hace en este ejemplo, con las llaves en líneas separadas y las instrucciones "indentadas" varios espacios. La razón es que, por su particular sintaxis, los programas en C tienden a convertirse en verdaderas sopas de letras difícilmente legibles. Para evitarlo, se han establecido este tipo de convenciones.

EL BUCLE WHILE

El concepto de bucle es clásico en programación. Es habitual que se necesite repetir una serie de veces alguna operación. La estructura while (mien tras) ejecuta una instrucción repetidamente mientras se cumpla una determinada condición. Cuando no se cumpla, el programa continúa normalmente. Tiene dos versiones, cuya sintaxis es la que sigue:

Formato 1:

while (<expresión>) <instruccion>

Formato 2:

do <instruccion> while (<expresión>);

En ambos casos se comprueba la <expre sión>, y, si resulta verdadera, se ejecuta la <instrucción>, se vuelve a evaluar la expresión, y así sucesivamente. La diferencia estriba en que la comprobación se haga antes (formato 1) o después (formato 2) de la instrucción. Como el lector habrá advertido, en el segundo formato la instrucción se ejecuta siempre al menos una vez; en el primero, al contrario, puede ser que no se ejecute nunca, pues la condición se evalúa antes.

while (x > 2) x=raiz_cuadrada(x);

En este ejemplo, x se sustituye por su raíz cuadrada repetidamente, mientras x sea mayor que dos. Si la primera vez no lo es, no se ejecuta nunca la función. Es corriente usar, como en el ejemplo, una variable a ambos lados se una asignación (=), con el objetivo de sustituir su valor por el resultado de aplicarle alguna operación. Igual que en if, las instrucciones deben terminarse con punto y coma (es una regla general en el lenguaje). Además, en el segundo formato, también se termina con punto y coma la cláusula while.

do x=x+1; while (x < 100);

En este caso, la variable x funciona como un contador (véase la siguiente sec ción), sumándole 1 mientras sea menor que 100. La suma se hará al menos una vez, pues la condición se evalúa al final. Por supuesto, en lugar de la instruc-

ción se puede siempre poner un bloque entre llaves:

```
do { y=x+1; x=y; } while (x < 100);
```

Este ejemplo hace lo mismo que el anterior, pero con más rodeos. Es costumbre poner el while en la misma línea que la llave de cierre, cuando va al final.

VARIABLES COMO CONTADORES

En la sección anterior veíamos un ejemplo de variable funcionando como contador, es decir, su valor va incrementándose, estableciéndose un límite como condición de terminación de un bucle. Su uso es muy común, normalmente para controlar un bucle que incluya instrucciones que haya que ejecutar un número determinado de veces. El contador controla el número de repeticiones que van hechas, hasta que se llegue a un límite.

```
i = 1; while (i < 10) { pitido(); i = i+1; }
```

La variable i se establece con un valor inicial y se incrementa hasta llegar a un valor final. Nuestro objetivo es que la función pitido() se ejecute 10 veces seguidas. Usar las letras i, j, k,... para designar los contadores es una "manía" que tienen muchos programadores. Por su puesto, cualquier variable puede hecer de contador, aunque es conveniente usar las de tipo int. Usar la expresión

```
i = i+1;
```

para incrementar obliga a acceder dos veces a la variable i para hacer una sola operación (incrementar). Esto puede ser un problema si estamos interesados en que el bucle se ejecute lo más rápido posible, lo cual suele ser casi siempre. Los inventores de C se dieron cuenta de ello, e incluyeron dos operaciones que no existen en casi ningún lenguaje (son típicas de ensamblador): el incremento (++) y el decremento (—). Estas operaciones se ejecutan mucho más rápido, pues sólo acceden una vez a la variable. Cada una de ellas tiene dos versiones: prefija y postfija, dependiendo si se ponen antes o después de la variable.

Pre-decremento: —variable
Post-decremento: variable—
Pre-incremento: ++varia ble
Post-incremento: variable++

Por ejemplo:

```
i++;
x—;
```

La primera instrucción incrementa en una unidad el valor de la variable i; mientras que la segunda decrementa en igual medida el valor de la variable x. Da igual el tipo de la variable; si es char se trata como int, a la manera habitual que ya hemos comentado. Como se ve en estos ejem plos, si la operación no forma parte de una expresión mayor, es decir, constituye por sí misma una instrucción, debe terminarse con el habitual punto y coma. Además, cuando se usa de esta forma el funcionamiento es el mismo en las versiones prefija y postfija. Las siguientes instrucciones funcionan exactamente igual; ambas incrementan en 1 la variable valor:

```
valor++;
++valor;
```

La diferencia aparece cuando se usan dentro de expresiones mayores:

```
x = ++dato;
```

Aquí hay dos operaciones: el incremento de dato y la asignación del valor de dicha variable a x. Pero, ¿qué valor asignar a x, el anterior o el posterior a la operación de incremento?. En el pre-incremento, primero se incrementa y luego se proporciona el valor para ser usado en el resto de la expresión.

```
j = 3; x = ++j;
```

Después de esto, ambas variables (x y j) valen 4. El pre-incremento ha sumado 1 a j, y después le ha asignado el valor ya incrementado a x. El efecto contrario se produce con el post-incremento:

```
j = 3; x = j++;
```

Tras la ejecución, j vale 4, mientras que x vale 3. El post-incremento proporcionó a x el valor de j antes de incrementarlo. Lo mismo ocurre en el siguiente ejemplo, aunque es un poco más difícil de ver:

```
dato = 16; x = raiz_cuadrada(dato—);
```

El post-decremento proporciona a la función el valor de la variable dato (16), y después le resta 1. Así pues, dato termina valiendo 15, mientras que la variable x vale 4 (la raíz cuadrada de 16, no de 15). Este ejemplo nos da una idea de lo "compactas" que puede ser las instrucciones en C: en una sola línea se han realizado tres operaciones diferentes (el decremento, la función y la asignación). Esto hace los programas más cortos y más rápidos, pero es más complicado de leer y entender. Si el lector se ha "atascado" un poco con este último ejemplo, no se preocupe: en las próximas entregas habrá oportunidades para explicar más ejemplos de este tipo de "instrucciones múlti ples". Una puntualización importante que, a primera vista, puede parecer de "perogrullo": el "objeto" al que se le aplique el incremento o el decremento deberá ser capaz de almacenar el resultado. Esto no siempre es tan evidente. Muchos principiantes intentan hacer cosas como éstas:

```
x = 3—;
x = (j+i)—;
```

Las dos instrucciones anteriores son totalmente incorrectas (no las acepta el compilador). En los dos casos se está intentando hacer un decremento a un objeto que no puede guardar el resultado. En el primer caso, dicho decremento se aplica a una constante (3) que, como es lógico, no puede cambiar de valor, ni tiene dirección en memoria donde almacenarlo (recuérdese que la asignación a x, aunque está relacionada, es una operación aparte del decremento). En el segundo caso, se intenta incrementar el resultado de una operación aritmética. El resultado de la suma de j e i no se almacena ni en j ni en i (sólo existe temporalmente para usar en la asignación a x), por lo que tampoco se puede almacenar el decremento. La operación correcta sería:

```
x = (j+i) - 1;
```

Todo esto es un poco complicado. Una regla fácil para recordar puede ser: Aplíquese el incremento o decremento sólo sobre objetos que pudieran ponerse a la izquierda de una asignación (=). Efectivamente, veamos la expresión siguiente:

(j+i) = 100;
Se ve fácilmente que es incorrecta. En el "argot" de C, se les llama "Lvalues" a este tipo de objetos. La L viene de "Left" (izquierda), es decir, "valores" que se pueden poner "a la iz quierda". De hecho, si el lector ha intentado compilar las instrucciones erróneas, habrá obtenido mensajes de error del tipo "Se requiere un Lvalue".

LA INSTRUCCION FOR

"For" es el otro tipo de bucle que puede hacer C. Esta instrucción es algo más extraña, pero mucho más flexible que las del mismo nombre en otros lenguajes.

for([<e1>]; [<e2>]; [< e3>]) [<instruccion>]

<e1>, <e2> y <e3> son tres expresiones opcionales separadas por punto y coma. La <instrucción> puede ser también un bloque de instrucciones entre llaves. Como siempre, los corchetes y los ángulos los hemos puesto con fines informativos, y no deben ponerse a la hora de programar. El funcionamiento es el siguiente: - Primero, se ejecuta <e1> (sólo la primera vez). - A continuación se ejecuta la instrucción (o el bloque) repetidamente como si se tratase de un bucle while ("mientras" la expresión <e2> sea verdadera). - Se añadirá al final la expresión <e3>, cada vez que se ejecute el bucle. Es decir, la instrucción for se "convierte" en:

<e1>; while (<e2>) { <instruccion> <e3>; }

Intentemos entenderlo con el ejemplo que usamos antes con while:

i = 1; while (i < 10) { pitido(); i = i+1; }

Todo esto se puede cambiar por:

for (i=1; i<10; i=i+1) pitido();

El resultado es el mismo, pero más compacto y más rápido de ejecutar. Junto a estas virtudes posee el defecto de ser menos legible. En estos casos, los programadores de C siempre prefieren la eficiencia a la simplicidad (por eso programan en C). Por cierto, ahora que conocemos el operador de incremento, podemos usarlo, mejorando aún más la velocidad de ejecución:

for (i=1; i<10; i++) pitido();

Esta instrucción ejecuta 10 veces la función pitido(), usando la variable i como contador. Da igual usar el pre-incremento o el post- incremento, pues la operación se realiza al final del bucle de forma separada. Este es, con diferencia, el uso más habitual del bucle for; pero, una vez conocido su funcionamiento interno, no es difícil imaginar decenas de formas de usarlo.

for (x=100; x>2;) x = raiz_cuadrada(x);

La tercera expresión se ha omitido (casi todo en for se puede quitar, pero no así los puntos y comas que separan las expresiones, ni los paréntesis que las delimitan). El bucle empieza dando a x el valor 100, aplicándole sucesivamente la función raiz_cuadrada() mientras se mantenga por encima de 2. A estas alturas ya poseemos la base suficiente para empezar a ver programas verdaderamente "ejecutables". El próximo capítulo lo dedicaremos a presentar varios ejemplos. Mientras esto llega, puedes intentar compilar y ejecutar el programa adjunto. (PONER AQUI LISTA.C)

No te preocupes si no conoces las órdenes #include y return, pues no hacen falta para entender el programa. La función putchar() pone un carácter en pantalla, cuyo código ASCII (tipo int) le proporcionamos. Intenta comprender cómo funciona el programa; ¿Qué pasa si quitamos el putchar('\n')? En el próximo número se explicarán con detalle éste y otros ejemplos.

```
/*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);              /*bucle for*/

  putchar('\n');

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

Figura 1

```
letra > 'A' && letra < 'Z'

raiz_cuadrada(x) >= logaritmo(y) - 1

letra_inicial != 100 && !(edad > 'A')

!(edad < 10 || edad > 70)
  && sexo != 'F'

(inicial == 'A'
  || inicial == 'B'
  || inicial == 'C'
  || inicial == 'X') && edad > 10
```

Figura 2

El C es un lenguaje secuencial, es decir, las instrucciones se ejecutan una a una, línea a línea.

El flujo normal del programa se puede cambiar a nuestro antojo mediante las bifurcaciones, las cuales están estrechamente vinculadas a las operaciones lógicas, mediante las cuales podemos hacer que nuestro programa actue de diferente manera dependiendo de el resultado de operaciones o simplemente de la toma de decisiones.

CURSO DE ENSAMBLADOR MOTOROLA 680000

Colaborador

Andrés Lopetegui

Nuestro nuevo colaborador nos introduce de lleno en las ventajas y los inconvenientes que encontramos a la hora de programar en código máquina en un ordenador Atari.

Las diferencias con respecto a la programación en un lenguaje de alto nivel son abismales y aquí te ayudamos a tomar tan importante decisión. Animate y sigue nuestro curso que ya va por su tercera entrega.

Muchos usuarios de Atari os estaréis preguntando cuales son las ventajas e inconvenientes de programar en Código Máquina. Una cosa es evidente, jamás podréis llegar a dominar un ordenador, sea del tipo que sea, si os dedicáis cómodamente a utilizar los lenguajes de alto nivel, como pueden ser el C, Pascal o Basic.

Desde nuestros programas, hechos en estos lenguajes, utilizamos funciones de librerías. Y muchos diréis: bueno, no entiendo que quiere decir eso de las librerías. Pues bien, voy a usar un ejemplo para interntar explicaros el problema, de forma que todos lo veáis claro.

En primer lugar vamos a intentar pensar que decidimos montar una empresa para fabricar coches. Una vez que hemos visto los pros y los contras del negocio finalmente nos lanzamos a la aventura.

Si pensamos un poco, hay dos formas de construir coches. La primera sería hacer nosotros los planos y después comprar las distintas partes en bloque. Así, compraremos a otros fabricantes el bloque motor, el bloque transmisión, el bloque chasis, el bloque tapicería, etc. Una vez que recibimos todos los componentes, procedemos a montarlos de forma que todas las partes de nuestro automóvil conjuntan a la perfección. Este sistema presenta varias ventajas. Por un lado no tenemos que preocuparnos de si las diferentes partes van a funcionar puesto que vienen con su garantía. Por otro, nos ahorramos mucho trabajo ya que no se requiere un esfuerzo de diseño de los componentes que compramos a otros fabricantes. El resultado, finalmente, es óptimo, pero nuestra idea de como se construye un coche es reducida. No sabemos, en ningún momento, como funciona el motor, o como de hace un chasis o transmisión, pero nos hacemos expertos montadores de automoviles.

Nos aprovechamos de los conocimientos de otros para hacer nuestro producto.

Trabajar con un lenguaje de alto nivel sería algo parecido. Creamos nuestro programa, que en el ejemplo anterior sería el coche, a base de piezas y componentes que otras personas han diseñádo previamente, que equivaldrían a las librerías a las que hicimos mención con anterioridad. De esta manera, se cae en la comodidad de la utilización de los lenguajes de alto nivel, alejándose el programador de la máquina con la que trabaja. Ello conlleva un riesgo bastante grande; cuando los fabricantes que nos tienen que suministrar las piezas fallan o cuando una pieza no encaja en

el conjunto y tenemos que retocarla para que así lo haga, no tenemos ni idea de como llevar ésto a cabo.

Una cosa quiero dejar bien clara y es que el uso de los lenguajes de alto nivel no es malo ni siquiera algo a lo que se deba rehuir, simplemente intento indicar que no se debe caer en su uso continuado y desmesurado por parte de todo aquel que quiera llegar a ser un buen programador de videojuegos.

La segunda opción de fabricación sería diseñar nosotros el coche de principio a fin y asumir también lo que sería la creación de todas las piezas que forman parte de nuestro producto final. De esta manera no se escapa de nuestras manos el proceso de fabricación el cual es supervisado por nosotros desde el primer momento. También adquirimos así los conocimientos y experiencia necesarios para mejorar nuestros automóviles y para diseñar muchos y variados prototipos.

La gran desventaja de este sistema es que tenemos que estudiar y trabajar más, pero por otro lado la gran libertad que ganamos con la máquina es maravillosa a la hora de soltar nuestra imaginación con los videojuegos y con cualquier otro tipo de programación.

La mejor opción de todas las que os he explicado es la que viene a continuación. Utilizar los lenguajes de alto nivel en conjunción con el código máquina o lenguaje ensamblador. Pero eso si, una vez que ya sabemos como hacer nuestros programas en bajo nivel. Conseguimos, de esta forma, ahorrarnos el trabajo donde no queremos caer en la rutina y acelerar la ejecución de nuestros programas cuando queremos obtener una máxima velocidad de ejecución.

Vosotros soís los que tenéis la última palabra. Comodidad frente a libertad y rapidez.

EL CURSO SIGUE

Nos quedamos en el número anterior justo en la explicación de los números binarios. Considero este tema de una importancia tan fundamental que prefiero tocarlo con más tranquilidad.

La notación binaria es fundamental para entender como funciona un ordenador por dentro y de como lleva a cabo su complicada tarea.

Aunque un ordenador os parezca algo inteligente, su sabiduría radica en una serie de corrientes eléctricas sincronizadas que dan vida y sangre a nuestro Atari. La existencia o inexistencia de estas corrientes (lo que se conoce de forma lógica como unos y ceros) permite tanto el almacenamiento de datos como el proceso de las órdenes por parte del microprocesador. Como véis los unos y los ceros son algo más importante de lo que pueda parecer a simple vista.

El microprocesador, también llamado CPU, que en el caso de nuestro ordenador lo fabrica Motolora y se conoce como 68000, toma de la memoria series de corrientes eléctricas y las interpreta como instrucciones o datos que es lo que configuran, én realidad nuestro programa. El resutado de esta operación es una actuación "inteligente" por parte de la CPU 68000, pero no nos llevemos a engaño. En realidad lo que el microprocesador hace es actuar automática y preprogramadamente a una serie de estímulo eléctricos y en función a lo que ha sido adiestrado a realizar, nada más. Por ello, nunca tomará una decisión humana o espontánea ante un problema que se le plantee, siempre echará mano a nuestro programa. Si este es incorrecto, así será el comportamiento de la CPU. Lo que nos lleva a pensar que en realidad es una pastilla de silicio muy tonta pero muy rápida.

Para que nosotros, los programadores, no tengamos que andar con voltajes de corrientes eléctricas, simplemente utilizamos sus equivalentes lógicos, los unos y los ceros que mencioné antes, los cuales podemos representar y trabajar con más facilidad.

El sitema binario tan sólo tiene dos números por lo que se le conoce como en base 2, mientras que el decimal (del 0 al 9) se le llama en base diez. A los unos y los ceros también se les conoce como BIT (Binary DigIT) o dígito binario.

Los números decimales se combinan mediante unas reglas preestablecidas para poder obtener número mayores de 10, por ejemplo el 143. Los números son siempre los mismos pero la posición que ocupan dentro de la cifra es lo que en realidad los dota de mayor o menor valor. El número situado más a la izquierda vale diez veces más (i.e. 2, 20, 200, 2000,...) que el situado a su derecha. Con los números binarios ocurre lo mismo. Cuando un 1 se coloca más a la izquierda, su valor aumenta, por ejemplo:

1	1
10	2
100	8
1000	16
10000	32

Véis como a medida que el número 1 se lleva más hacia la izquierda el número binario pasa a valer el doble que valor que tenía en la posición situada a la derecha.

Para averiguar el valor de cada posición binaria elevamos el número dos por la posición en sí.

Así tenemos:

Posición 0...$2^0=1$
Posición 1...$2^1=2$
Posición 2...$2^2=4$
Posición 3...$2^3=8$
Posición 4...$2^4=16$
Posición 5...$2^5=32$

En el próximo número veremos porque esto es así.

curso de ensamblador Motorola 68000

CODIGO JUEGO

AMIGA

Colaborador

Carlos Garrido Novoa

Comenzamos este mes con un curso para los que gustan de la programación a toda potencia en el ordenador AMIGA. Lo prometido es deuda, así que Carlos, nuestro amigo en la materia, empieza, este mes, con la primera entrega de lo que llegará a ser un lugar donde encontraréis un apoyo para dar velocidad a vuestros juegos

¡Hola a todos!

Esto que comienzas a leer es un curso de programación en ENSAMBLADOR para la familia de Microprocesadores 68000 de MOTOROLA, especialmente indicado para el ordenador Amiga, pero si eres poseedor o usuario de un Apple o un Atari y quieres aprender a programar este fabuloso micro, te aconsejo que sigas este curso junto a un buen libro de programación en ENSAMBLADOR para tu ordenador, ya que se complementarán.

Referente a este curso, quiero que tengas en cuenta que vamos a comenzar desde cero absolutamente, como si fuera la primera vez que enciendes tu Amiga, (no será el caso, ¿¿¿verdad ???...), aunque si así fuera no habría ningún problema.

El curso te guiará poco a poco por este maravilloso y en muchos casos satisfactorio mundo de la programación, hasta alcanzar un nivel de auténtico profesional, pudiendo realizar cualquier cosa que se te ocurra, teniendo como único límite la capacidad de tu imaginación, pudiendo desarrollar desde una sencilla base de datos hasta el algoritmo más complicado que mueva un muñeco a lo largo de un sinfín de niveles en tu propio juego.

Este curso tendrá como ejemplo el Amiga 500 y 2000, por ser estos los ordenadores Amiga más extendidos, pero prácticamente la totalidad de lo que estudiemos se podrá aplicar en el resto de la familia Amiga.

Esperamos que nadie se raje por los comentarios que continuamente se mueven de que el LENGUAJE ENSAMBLADOR, es muy difícil y sólo para eruditos sesudos, ya que yo soy de lo más normalito y no me ha supuesto ningún transplante de cerebro aprenderlo y sacarle el máximo partido, al igual que vosotros si
os lo proponéis y seguís un paso ordenado como el que os proponemos en este curso y no os saltáis ninguna parte por fácil que parezca.

Ahora aprenderemos un poco de teoría básica para diferenciar los distintos niveles de programación.

COMENZAMOS

El microprocesador, desde que se enciende el alimentador, necesita una serie de instrucciones que le digan lo que tiene que hacer.

El microprocesador tiene unas patillas a las que le llegan unas pistas, (estas pistas son como cables). Por cada una de éstas puede no pasar corriente, a este estado se le llama cero (0), o puede pasar corriente, lo que se denomina uno (1), y ya no existen más casos. Que no se os pase por la cabeza la posibilidad que me daba vueltas a mi cuando estaba aprendiendo: y si pasa la mitad de corriente, ¿quizás sea el estado 0.5 ?, un rotundo ¡¡NO!!.

La concatenación de pistas (unir una detrás de la otra), produce lo que llamamos buses, que son 32 de estas pistas y que pueden estar todas a cero, a uno, o una variación de unos y ceros que, a los humanos y a simple vista, no nos dicen nada, pero para los microprocesadores es absolutamente comprensible. Además, interpretar este infromación para ellos es una tarea que deben realizar en el acto, por ejemplo:

1101000001111100
0000000000000110

significa una orden que debe cumplir el microprocesador y que traducida al lenguaje humano quiere decir:

SUMA EL NUMERO 6 AL REGISTRO D0 CON UN RANGO DE PALABRA Y EN MODO NORMAL.

No te asustes puesto que esto no tiene nada de complicado y no te lo tienes que aprender, yo tampoco lo sé, simplemente me limito a mirar tablas.

Si sólo una de estas pistas cambiara su valor, (en vez de un 0 un 1 ó viceversa), la orden sería distinta, por ejemplo:

1101000001111100
0000000000000111

y ahora su significado seria:

SUMA EL NUMERO 7 AL

REGISTRO D0 CON UN RANGO DE PALABRA Y EN MODO NORMAL.

Si tenemos esto en cuenta nos percataremos de que hay millones de combinaciones posibles y a no ser que fuéramos 150 elefantes superdotados juntos, es del todo imposible programar así, ademas de muy aburrido y nada gratificante, pero sólo una consecución de estas ristras de ceros y unos kilométricas es lo que
el microprocesador entiende y devora a cada nanosegundo.
Como los programadores hace mucho tiempo se dieron cuenta de que es imposible este método de programación, diseñaron un lenguaje que fuera más comprensible para los humanos y a la vez no se alejara mucho de la forma de trabajar del microprocesador; el resultado fue el LENGUAJE ENSAMBLADOR,
en el cual ES POSIBLE PROGRAMAR SIN PERDER EL CONTACTO CON EL MICRO.
Para poner un ejemplo del aspecto de este lenguaje, estas son las dos instrucciones que hemos comentado anteriormente, pero esta vez en este otro lenguaje:

ADD.W #6,D0

ADD.W #7,D0

Esta forma de programar, como se puede apreciar, es más cómoda (y más todavía para los ingleses que como son ellos los inventores, pusieron sus propias palabras como ADD que significa SUMAR), pero aun así es muchísimo mas normal y aceptable.
Y finalizada esta pequeña introducción, vamos a dar paso a la siempre controvertida pregunta...

¿POR QUE LENGUAJE ENSAMBLADOR ?

Como acabamos de aprender, el LENGUAJE ENSAMBLADOR es lo más parecido al Código Máquina o Lenguaje propio del microprocesador , en la escalera de la programación, pero como todos sabemos existen muchos otros lenguajes como el BASIC, C, COBOL, PASCAL, etc. En primer lugar, sólo por ser el más cercano al lenguaje del MC (micro), tenemos la ventaja de tener un mayor control sobre todo el ordenador,
ventaja más que suficiente para decantarse por este lenguaje, pero además de ésta existen otras muchas, que analizaremos aquí ahora.
Un punto que debemos tener claro es que cuanto más cerca estamos del lenguaje del MC, más difícil es aprender dicho lenguaje, pero como compensación obtendremos una grandísima velocidad, un programa muy compacto y muy corto y además, podremos programar todo lo que nuestro ordenador pueda realizar, pudiendo realizar cosas que en otro lenguaje sería imposible.
Por otra parte, cuanto más nos alejemos del CM, obtendremos un lenguaje mucho mas fácil de programar (el BASIC, por ejemplo), pero más lento, menos compacto, y mas largo resultará nuestro trabajo, y con muchísimas limitaciones. Por este motivo muchos programadores aprenden dos lenguajes y los intercalan
utilizando el ENSAMBLADOR en la mayor parte del programa, sobre todo en aquellas en que la velocidad y sincronización sea crucial, y otro lenguaje (que suele ser C), en las partes en que no es precisa ni mucha precisión ni velocidad, siendo este el recurso que se suele utilizar a la hora de programar cualquier cosa.
Todo esto esta muy bien y dice mucho a favor del ENSAMBLADOR, pero, ¿hasta qué punto pueden ser tan necesarias la velocidad o la longitud del programa?, pues si nosotros pretendemos realizar un juego con 6 fondos, 4096 colores simultáneos (si, es posible), música, muñecos a todo trapo (sprites, bobs, etc..), vectores 3D, etc. ni se te ocurra hacerlo en BASIC1, puesto que el resultado sería verdaderamente deprimente después de las muchisimas horas de programación que necesitaríamos y obtendríamos un muñeco que tardaría media hora en pasar de un lado de la pantalla al otro. La jugabilidad y adicción serian de -100, aparte de necesitar
120 discos para almacenarlo.
El C, sin embargo, es mucho mas útil para realizar este tipo de programas, aunque si lo que se persigue es seriedad y profesionalidad la elección es el rapidísimo LENGUAJE ENSAMBLADOR, que resulta unas 300 veces mas rápido que el BASIC, y unas 25 mas que el C. Si a esto se le suma que el programa en ENSAMBLADOR ocupa muy poca memoria en relación al resto, obtendremos el motivo de porque
el 90% de los programas y juegos comerciales están realizados en ENSAMBLADOR.
La principal desventaja del ensamblador es su dificultad, ya que se necesita conocer perfectamente como trabajan todas las tripas del ordenador, disponer de numerosas tablas de memoria, conversión, comandos, etc., pero una vez asumido esto es realmente satisfactorio, llegando incluso a convertirse en
tu hobby favorito, como es mi caso.
Dicho esto espero que no te quede ninguna duda del por que de la inclinación de los programadores hacia este
lenguaje.

SISTEMAS DE NUMERACION:

Todo buen programador debe conocer los sistemas de numeración más usados en el campo de la
programación, los cuales son:

- BINARIO.
- OCTAL.
- DECIMAL.
- HEXADECIMAL.

Vamos a comenzar analizando el sistema decimal por ser el que conocemos y del que menos nos costará
sentar sus bases. El sistema octal no lo vamos a estudiar por estar en desuso, simplemente lo incluyo para que
te suene y te sea familiar el nombre.

SISTEMA DECIMAL

Este sistema es el que utilizamos en la vida cotidiana, y con el que pensamos los humanos. Este sistema tiene
como base el 10, esto quiere decir dos cosas:

1- En este sistema existen 10 dígitos distintos (0,1,2,3,4,5,6,7,8 y 9), teniendo un valor distinto cada uno de ellos.
2- El 10 es el menor número con el cual debemos correrlos un puesto en el número (hacia la izquierda).
Vamos a aclarar esto. ¿Cómo representas el cero en decimal ?

```
------
|||||0|
------
```

¿y el numero inmediato superior?

```
------
|||||1|
------
```

¿y el numero inmediato superior?

```
------
|||||2|
------
```

¿y el numero inmediato superior?
y así hasta el

```
------
|||||9|
------
```

entonces, el inmediato superior será...

```
------
||||10|
------
```

Pues el 10 es menor numero que en una casilla no es posible meter.

En el próximo número continuaremos con el sistema decimal y el resto de las notaciones numéricas.

Profesor

José Manuel Ginés

Un mes más el curso de MIDI llega a vosotros de la mano de nuestro experto en la materia. Pronto sabremos lo suficiente para empezar a ver el formato de los ficheros .MID con los que podremos manejar la música desde nuestros juegos. Continua el estudio de las características del interface con el que podrás dominar la melodías más increíbles.

CURSO DE MIDI

Musical Instrument Digital Interface

El mes anterior comenzamos el estudio de los mensajes MIDI. Retomamos el estudio en el punto en que lo dejamos cuando hablábamos de los mensajes de selección de canción.

Hoy en día con el empleo de secuenciadores y cajas de ritmo arriba del escenario este mensaje es sumamente valioso. Una canción en un secuenciador es el conjunto de secuencias o patrones que son ordenadas secuencialmente para crear una composición completa. Los instrumentos que pueden transmitir o recibir este mensaje, emvian esta instrucción para que los dispositivos esclavos se cambien a la siguiente canción. El usuario decide el orden de las canciones.

(C) AFINACION

Cuando el Midi fue incorporado a los sintetizadores, la mayoría eran analógicos y no digitales.
Los sintetizadores analógicos trabajan por medio de osciladores controlados por tensión o VCO's. Los osciladores son muy inestables y se desafinan constantemente, por esa razón los fabricantes, optaron por incorporar un botón en el panel frontal del sintetizador que lo llamaban (tune). Bien, el mensaje de afinación es transmitidos con el propósito de afinar los sintetizadores analógicos a control remoto vía controlador maestro. Esto es utilizado en una actuación en vivo cuando en la canción que se esté tocando, el sintetizador analógico no se está usando, se le puede enviar el mensaje de afinación para que esté afinado al momento en que se utilice. Por supuesto ésto es únicamente posible si éstos aceptan este mensaje. Hoy en día la mayoría son digitales y no necesitan afinación.

(D) FIN DEL SISTEMA EXCLUSIVO

Este mensaje es enviado por el transmisor o controlador maestro para hacer

saber al dispositivo esclavo que la transmisión del mensaje de sistema exclusivo ha sido terminada.

(2) MENSAJE DE TIEMPO REAL

La manera de sicronizar instrumentos electrónicos como sintetizadores, pianos electronicos, secuenciadores, etc... antes de que el Midi existiera era por medio de pulsos eléctronicos llamados pulsos de reloj. Estos eran usados para sincronizar un secuenciador con una caja de ritmo donde uno de ellos actuaba como el reloj maestro y el otro como el reloj esclavo y se conectaban de la siguiente manera:
Si el secuenciador era el maestro entonces se hacía la conexión con el conector llamado salida de reloj, hacia el conector llamado entrada de reloj, de la caja de ritmo por medio de un cable de audio, es decir un cable para guitarra.
La sincronización hecha de esta manera funcionaba muy bien, excepto que existía un problema, que no todos los fabricantes de sintetizadores utilizaban la misma velocidad de reloj. La velocidad de reloj es el número de 4/4 existen cuatro notas negras, así , si la velocidad de reloj es de 96 pulsos o tiempos por cada nota negra que pasa, entonces habría 384 tiempos en un compás de 4/4.
La compañia de sintetizadores Oberheim usa la velocidad de reloj de 96, otras compañias usan 48, 24, 64, 384, pulsos por cada nota negra. Ya se puede dar cuenta lo problemático que era entonces el poder sincronizar secuenciadores y cajas de ritmos de diferentes marca, pero el Midi alivia este problema. La MMA estandarizó a lo que se le da el nombre de reloj de 24 ppnn. Todas las compañias de sintetizadores optaron por incorporar ya a sus secuenciadores y cajas de ritmo el reloj Midi, para ser generado y recibido, de esta manera es más sencillo y eficiente poder sincronizar aparatos de diferentes marcas.
Los mensajes de tiempo real se refieren a la capacidad de instrumento o aparato Midi para que pueda sicronizarse con otros. Tiempo real significa que uno va a percibir el cambio de una acción espontáneamente, por ejemplo, el combio instantáneo de un sonido en el momento en que se efectuó éste. Los mensajes de tiempo real no se envían por canales Midi especifícos, se dividen en:

(A) TIEMPO DE RELOJ

Cuando un secuenciador o caja de ritmo actúa como maestro,éste transmite el mensaje de tiempo de reloj. A propósito, estos mensajes únicamente consisten en un byte que es el de estado. Los receptores en este caso son los sintetizadores esclavos y cajas de ritmos. Estos recibirán el código F8H (tiempo de reloj) en su entrada Midi. Existen en el mercado sintetizadores que contienen un secuenciador interno. Un ejemplo de éste es el O1 de Korg. Este tipo de teclado puede aceptar el mensaje de tiempo de reloj. El tipo de sintetizadores que no contienen secuenciadores internos, simplemente ignoran este mensaje. Cuando el secuenciador, en este caso el controlador maestro está en estado de reposo, seguirá enviando mensajes de tiempo de reloj. Esto es para que al momento de que empiece el maestro a tocar las secuencias que contiene en su memoria, la caja de ritmo y los sintetizadores esclavos tengan oportunidad de calcular el tiempo en que éstos deben empezar a tocar en sincronización con el secuenciador. Debo mencionar que la mayoría de secuenciadores y cajas de ritmo tienen una función llamada sincronización interna, y otra llamada sincronización externa. Esta función es la que decide quien va a ser el controlador maestro. Si por ejemplo el secuenciador es el maestro, entonces la función que debe seleccionarse en el secuenciador es el de sincronización interna. Esto quiere decir que el secuenciador va a trabajar con su propio reloj, po lo contrario, el reloj del secuenciador, consecuentemente, la caja de ritmo debe estar en la función de sincronización externa.

(B) MENSAJE DE INICIO

Cuando el botón de inicio del secuenciador o caja de ritmos, se oprime, el mensaje de inicio (FAH) se transmite al igual que el mensaje de tiempo de reloj, y sirve para empezar a tocar un secuenciador o caja de ritmos a control remoto desde el principio de un fragmento musical ya programado en sus memorias. Los instrumentos esclavos tocarán a la misma velocidad que el compás del maestro.

(C) MENSAJE DE PAUSA

El mensaje de pausa sirve para proseguir el fragmento musical desde el punto donde fue detenido, en otras palabras que no va a empezar desde el primer compás de la composición. La pausa es de gran utilidad ya que nos permite parar durante un tiempo sin que por ello tengamos, necesariamente que escuchar la composición entera al reanudar nuestro trabajo.

(D) MENSAJE DE TERMINACION

El mensaje de terminación consiste en detener el fragmento en el punto deseado desde el controlador maestro.

(E) SENSOR ACTIVO

Este mensaje se incorporó en la especificación Midi con el propósito de evitar las notas estancadas que comunmente tenemos al desconectar el cable Midi durante la transmisión de información Midi. Una nota estancada en un sintetizador es cuando el sonido de éste queda permantemente activado. Cada nota activada en el sintetizador,, debe seguirle una desactivada. Ahora, si cuando se desconectó el cable Midi y el último mensaje que recibió el sintetizador esclavo fue el de una nota activada, entonces éste no va a recibir el mensaje de la nota desactivada porque la

transmisión cesó en ese instante; consecuentemente hasta que el sintetizador se apague, o bien, hasta que se envíe el mensaje de desactivación total de sonido. El controlador maestro lo transmitirá cada 300 milisegundos, o sea cada tercia de segundo, para hacerle saber al esclavo que todavía existe una conexión y que la línea de transmisión está abierta sin problema alguno. Ahora, en cuanto el receptor esclavo deja de recibir este mensaje, después de que haya pasado un lapso de 300 milisegundos, este asumirá que el cable Midi se desconectó, e inmediatamente cerrará las compuertas de transmisión y apagará todos los sonidos que estaban activados.

(F) RESTABLECIMIENTO DE SISTEMA

Este mensaje generalmente es enviado por el secuenciador maestro hacia los sintetizadores esclavos para restablecer los parámetros de los sintetizadores a los valores que tenían cuando se les aplicó energia, es decir, desde el momento en que estos se encendieron por primera vez. Por lo general los aparatos vienen asignados ya desde la fábrica a diferentes funciones. Digamos que al encender el sintetizador, éste está asignado para que reciba información por todos los canales de Midi. Esto significa que la función OMNI esta activada; también dejamos que la función local viene activada, etc... para estar seguro que están funcionando, desde el maestro puede enviarse el mensaje de restablecimiento de sistema para asegurarse.

(G) MENSAJE DE SISTEMA EXCLUSIVO

Los mensajes de sistema exclusivo que comunmente son llamados Sysex, son usados para enviar información de un dispositivo Midi de determinada marca a otro dispositivo de la misma compañía y éste debe ser el mismo modelo de aparato. Por ejemplo, en el caso del sintetizador DX7, si deseo enviar todos los programas o sonidos de un DX7 a otro Dx7 lo podremos hacer, pero si quiero enviar todos los sonidos de un DX7 a un D-50 de la compañía Roland, no será posible. Esto es porque el Dx7 fue diseñado con la tecnología de frecuencia modulada y el D-50 con la tecnología llamada llamada Aritmética lineal. Por lo tanto, si envío parámetros como los operadores del DX7 al D-50 , el D-50 no va a entender lo que es un operador porque no fue diseñado con tecnología de FM.
Los mensajes de sistema exclusivo contienen un byte de estado (FOH) y varios (más de dos) bytes de información. Noten la diferencia de bytes de información del resto de los mensajes que hemos estado hablando , ellos únicamente son enviados con un máximo de tres bytes . La razón por la cual existen más byte de información en los mensajes de sistema exclusivo, es porque la información que se envía por medio de estos mensajes está formada por los parámetros de los sonidos. ¿ Recuerdan cuando hablé del mensaje de cambio de programa? mencioné que el mensaje de cambio de programa únicamente envía el número de programa del sonido y no los parámetros (filtros, VCO's, generadores envolventes, etc.) de éstos. Por acuerdo de la MMA y la JMSC (comité Japonés Estandarizador de MIDI), a cada manufacturera se le asignó un número de identificación. Este número es exclusivo para cada fabricante, es decir, no puede haber dos números de identificación iguales. La razon es porque al enviar el mensaje de sistema exclusivo a un dispositivo esclavo, si el número enviado en el mensaje no es el mismo que el del dipositivo esclavo, éste ignorará el mensaje y no aceptará ninguna información, esperará hasta que reciba el mensaje de fin de sistema exclusivo (F7H) que es un mensaje de sistema común. Este mensaje hace saber al dispositivo esclavo que la transmisión de información ha sido completada. El orden en que un mensaje de sistema exclusivo se envía es el siguiente:

1) Como se estableció anteriormente el primer byte que se envía es el de estado que en este caso es el FOH en forma hexadecimal. Cada vez que se encuentre este número (FOH) en un manual de instrucciones significa que es un mensaje de sistema exclusivo.

2)El segundo byte enviado, es el primer byte de información y es el número de identificación del fabricante.

3)Sigue el número de canal de MIDI (del 1 al 16) en que el receptor aceptará el mensaje.

4)En seguida, se envía el modelo del aparato. Por ejemplo si es un DX7, un D50 o un Matrix 1000 de Oberheim. Estos datos son informativos, no es algo sumamente necesario para utilizar Midi.

5) El resto de los bytes de información incluyen lo que se está transmitiendo, ya sean los parámetros del propio sintetizador o las muestras, o bien las secuencias de una caja de ritmos o secuenciador.

6) El último byte enviado es muy importante y se le da el nombre de byte de fin del sistema exclusivo que sirve para hacerle saber al receptor que la transmisión fue totalmente realizada.

Todos los fabricantes de dispositivos con Midi, estan obligados a publicar al final de cada manual de instrucciones o libros de texto técnicos su número de identificación al igual que la definición de cada parámetro. Esto es para que las compañías que se dedican a elaborar programas puedan llevar a cabo su proyecto, lo cual es beneficioso para la compañía manufacturera de dispositivos Midi. Por lo general éstas compañías lanzan al mercado secuenciadores en programa para usarse en

los ordenadores personales, otras se dedican a escribir programas para edición, archivo y organización de sonido. Ejemplos de estas compañías son Digidesign, Opcode y Mark of the Unicorn. Algunas de las aplicaciones del mensaje de sistema exclusivo son: El enviar sonido de un dispositivo con Midi a otro de la misma marca. Por ejemplo, digamos que un teclista sale a una gira mundial con un cantante famoso. El debe llevar por lo menos dos sintetizadores de cada modelo, idénticamente programados, para evitar riesgos en caso de que uno de ellos falle durante el concierto. En éste caso, el teclista tiene que copiar los sonidos de un sintetizador a otro, ya que le costaron horas y horas de trabajo y no puede programarlos, en ese instante. Bien, en lugar de que tenga que programar desde el principio todos los sonidos por medio de los Mensajes de Sistema Exclusivos, no tendrá que reprogramar el sintetizador extra, la mayoría de éstos y demás dispositivos Midi hoy en día, pueden enviar Mensaje de Sistema Exclusivo. En el panel frontal de cada dispositivo se en cuentra un botón llamado Midi. Por medio de este botón, el usuario podrá enviar sonidos, secuencias, muestras vía los Mensajes de Sistema Exclusivo. Debo mencionar que no todos los dispositivos pueden enviar los mensajes desde su panel frontal. En caso de que ésta sea la situación, el usuario tendrá que adquirir un dispositivo llamado Archivo de información como el DataDisk , que tiene la capacidad de solicitar el envío de información (sonido, secuencia, sampler) desde un dispositivo a éste y poder así mandar la información a su destino. Otra alternativa es adquirir el programa apropiado para llevar a cabo esta función y utilizarlo por medio de el ordenador. En el siguiente caso, la aplicación de los mensajes de Sistema Exclusivo (sysex) es para editar y organizar sonidos visualmente con la ayuda de un ordenador personal. A esta clase de programa se les da el nombre de Editor de sonidos y Archivo de sonidos, patch editors y sound librerian, respectivamente. Existen sintetizadores sin teclado llamados módulos generadores de sonido que pueden ser controlados por un teclado maestro. Tomemos como ejemplo el Matrix 1000 de la compañía Oberheim. Este módulo contiene mil programas de los cuales ninguno puede ser editado y alterado por medio del panel frontal de éste. El problema con este módulo es que en el caso de que uno encuentre el sonido casi perfecto para una pieza de música o efecto de sonido para un comercial, no podrá alterarlo para obtener el sonido al gusto. Para ello existe un programa que facilita la edicción de los sonidos de los módulo, utilizará los mensajes de sistema exclusivo para enviar los sonidos desde el Matrix 1000 a el ordenador para así poder editar los sonidos deseados.

Hoy en día los estudios Midi domésticos, es decir, estudios en casa, son muy usuales. Los precios de los instrumentos con Midi han decrecido tanto que le es posible a cualquier persona poder adquirir el equipo necesario para montar un estudio de grabación en su propia casa. El uso de secuenciadores es muy común entre los compositores y sirven como una herramienta para grabar todas las ideas musicales en tiempo real. Existen ocasiones en que el compositor tiene sus obras en un secuenciador en programa y desea llevarlo a un estudio de grabación profesional para ser grabado y mezclado y poder lanzarlo al mercado en un disco de vinilo o en un disco compacto que es de más alta fidelidad. Digamos que el compositor también posee un secuenciador físico, en hardware. Para no tener que llevarse el ordenador al estudio de grabación, éste puede, por medio de mensajes de sistema exclusivo, transferir las seculencias desde el ordenador hacia el secuenciador físico.

Las cajas de muestreo comunmente llamados SAMPLERS utilizan también los mensajes de SYSEX. Un ejemplo de éste es cuando el usuario necesita enviar muestras de un SAMPLEADOR a otro, digamos de un EMULATOR III a un ADS (Advance Digital Sampler). La transmisión de las muestras se efectúan por medio de lo que se llama estándard del vaciado de muestras universal, universal sample dump, que por acuerdo de la MMA y JMSC es un estándar para poder enviar muestras o sonidos de un SAMPLEADOR a otros de distintas marcas. Existen alglunos SAMPLERS como el K2000, S900 de Akai que tienen su propio código para el envío de muestras. Para el intercambio de muestras entre el S900 y otro SAMPLEADOR se necesita un programa para que lleve a cabo el intercambio. La compañía Digidesign lanzó al mercado el programa llamado UNIVERSAL SOUND DESING que puede translladar muestras de cualqulier tipo de Sampler al ordenador (trabaja con Atari y Macintosh). Estando la muestra o sonido en el ordenador el usulario tiene la facilidad de editarlo ya sea que ponga el principio del sonido al final o en la parte media o para crear efectos especiales. Ya que el sonido es editado, éste se envía vía Midi por medio de los mensajes de Sistema Exclusivo hacia su destino, es decir, hacia el SAMPLER deseado sin tener que preocuparse, éste respeta el vaciado de muestras universal.

Existen una inmensa variedad de aplicaciones para el Sistema Exclusivo, pero éstas son las más comunes.

El próximo mes continuaremos. ¡Ah!, recordad lo que os dije el mes pasado al final de mi artículo.

Ya queda poco para que haya música a raudales en tus juegos.

Microsoft Macroassembler 6.0

Analisis y estudio

Director de CJ

Ramón López

El mes pasado empezamos a analizar este producto de la compañía Microsoft, la cual no requiere presentaciones de ningún tipo, ya que es de todos conocida por la calidad de sus productos y principalmente por sus dos estrellas; el sistema operativo MSDOS y el entorno operativo WINDOWS, sobre el cual comenzamos un curso en este mismo número.

El MASM, o Microsoft Macroassembler, es un paquete de programación en lenguaje ensamblador que reúne todos los componentes necesarios para llevar a cabo aplicaciones profesionales en dicho lenguaje.

Ya el mes pasado desmenuzábamos algunos de ellos, como PWB, a la vez que analizábamos sus características esenciales.

El lenguaje ensamblador, y sobre todo el método en el que había que trabajar con los programas creados utilizando este lenguaje, resultaba tosco y rudimentario hasta extremos imperdonables. En primer lugar, los listados fuente, es decir los archivos de texto que nosotros los programadores escribimos y que conforman realmente los programas, habían de ser creados utilizando un procesador de textos de una tercera firma o por el contrario utilizar el sencillo EDLIN, que ni mucho menos ofrece ventajas y sencillez en la edición de textos.

Usar un editor no integrado con el compilador nos llevaba a un proceso como el de la figura adjunta.

El programa escrito y corregido sintácticamente ha de ser guardado. Tras ello se sale de la aplicación que lo creo (ya sea Wordperfect, Word, Edlin, Edit, etc.) y entonces, desde la línea de comandos del DOS se realiza todo el proceso de compilación con MASM y enlazado con LINK. En aquellos casos en los que queramos que nuestro programa adopte un formato .COM en lugar de .EXE, todavía deberemos realizar un tercer paso con la utilidad EXE2BIN. Cualquier fallo en la compilación requerirá que demos un salto al paso 1 y vuelta a empezar.

Este sistema presenta dos claros handicaps. Por un lado el hecho de tener que abandonar el editor y realizar la compilación mediante complicadas líneas de comandos que nos conducen a más errores de los que desearíamos. En segundo lugar, la cantidad de archivos intermedios con los que llenamos nuestro disco duro (.ASM y .OBJ) y que son el resultado de muchas compilaciones seguidas, en la mayoría de los casos erróneas.

Los ensambladores de la nueva generación, como el que es motivo de nuestro comentario, resuelven este problema mediante la sencillez y la integración de tareas. Desde dentro del mismo editor conseguimos llevar a cabo no sólo compilaciones completas, sino hasta depuraciones del código resultante de nuestro trabajo.

Este proceso, llamado BUILD (construcción), aúna todos los pasos de compilación y enlazado que antes debían realizarse desde la línea de comandos del MSDOS. Se han sustituido las opciones del tipo "/s /2 /stack:1234" y otras parecidas, por menús de persiana donde con un simple "click" de ratón logramos establecer la configuración de todas nuestras compilaciones, pudiendo, incluso, guardar toda la información sobre las mismas bajo un archivo y con un nombre que nos recuerde el programa del que se derivan. Nos ahorramos de esta forma mucho

trabajo y muchos errores. Nada mas que lo explicado, ya haría que mereciese la pena adquirir un MASM, pero no dejéis de leer, aún hay más.

BROWSE

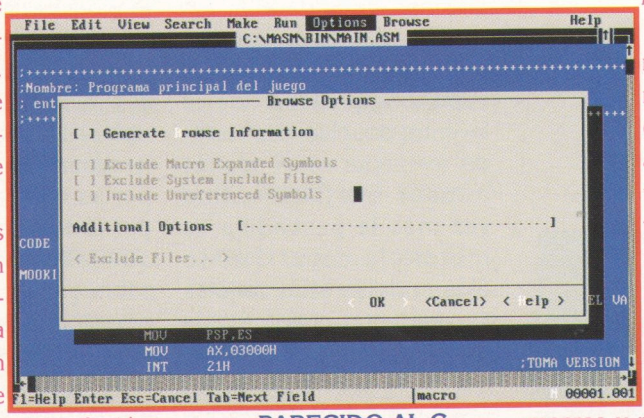
Este término, que significa "echar un vistazo", "ojear", reúne un potente sistema de visualización del comportamiento de las variables, funciones, parámetros y etiquetas que hemos creado dentro de nuestro programa. No es extraño, sobre todo en programas muy largos, que nos encontremos con problemas en el reconocimiento de nuestras variables, así como a que módulo pertenecen. Especialmente al utilizar código fuente que otros crearon.

Si activamos la opción "crear información para BROWSE" en el menú de opciones de la compilación, el ensamblador generará los datos necesarios para que se pueda encontrar la relación entre las variables y las partes del programa donde fueron creadas.

BROWSE permite no sólo tener control de esta relación, sino también comprobar el árbol de nuestro programa, ver que funciones o módulos hacen uso de ellas, ver su historial, comprobar el lugar donde se definieron, etc. Esto se hace especialmente útil cuando, por ejemplo, queremos ver cuantas llamadas se hacen a una subrutina, o cuando deseamos eliminar una variable sin que por ello conduzcamos nuestro programa hacia el profundo y oscuro pozo de la desesperación.

BROWSE también es excepcional a la hora de ver cuantos macros hemos creado y cual es la función de cada uno de ellos.

En resumidas cuentas, con esta utilidad, que al igual que las demás está integrada en el entorno del PWB, no ayuda a seguir el transcurso o árbol de nuestros programas para evitar que nos nombres de las etiquetas que hemos creado, lleguen a perdernos en la incertidumbre.



PARECIDO AL C

Con el MASM 6.0 y 6.1 lo que se ha intentado es hacer del lenguaje ensamblador una herramienta de programación apetecible. Se pensó, a poco que salió a la luz el PC en 1981, que este lenguaje resultaría muy difícil y poco atractivo para el usuario medio. Derivó esto en una falta de atención total por parte las compañías creadoras de este tipo de aplicaciones, las cuales olvidaron a los ensambladores dedicando más tiempo y esfuerzo a la mejora de los compiladores de otros lenguajes más comerciales como el C o el PASCAL. Pero mira por donde, al contrario que tornarse un lenguaje arisco, los programadores de alto nivel se dieron cuenta que conocer el ensamblador podía ser una gran ayuda en ciertos momentos en los que la programación de estos lenguajes sofisticados resultaba lenta o ineficaz.

En ese preciso instante se demandó un sistema de programación profesional que pudiese cumplir con las exigencias de cualquier creador de software y compañías como Microsoft y Borland, finalmente, sacaron al mercado sus productos; el MASM y el TASM.

Se ha logrado no sólo un sistema "user friendly" (o "apetecible o amistoso para el usuario"), sino también una forma de programar muy parecida al lenguaje C. Mediante el uso de potentes macros y nuevas instrucciones y directivas de ensamblador ahora, mejor que nunca, programar en ensamblador es cosa de niños.

Gracias a estas ayudas, ya no podemos negarnos a programar con el único lenguaje que logra sacar partido totalmente de nuestro ordenador.

De la utilización de todos y cada uno de los menús de PWB, así como la forma en que debemos compilar un programa ejemplo, hablaremos en el próximo número de Código Juego.

De lo que se trata es de analizar los diferentes ensambladores profesionales que existen en el mercado del PC. Por ello, cada mes, en este apartado de Código Juego, os ayudaremos a escoger cual es el que más os interesa, así como explicaremos como trabajan y cuales son sus pros y sus contras.



CODIGO JUEGO



Curso

Los mil y un sonidos del PC

Profesor

Ramón López

Una cosa, de las pocas diría yo, que si tenía el PC cuando nació era la capacidad de producir sonidos; rudimentarios, eso sí, pero sonidos en el fondo. Un pequeño BIP era lo único que obteníamos y, por lo general, al equivocarnos introduciendo datos en la hoja de cálculo.

Los pobres programadores de videojuegos, nuestros ancestros, se las veían y se las deseaban a la hora de crear efectos de sonido que adornasen de forma acompasada y con calidad, la acción que se estaba desarrollando en la pantalla del ordenador.

La verdad es que en la mayoría de los casos el efecto conseguido no pasaba de ser un ridículo zumbido, que más que ayudar a la calidad del juego, lo que conseguía era recordarnos que estábamos frente a una máquina y no jugando con algo casi real.

Otros ordenadores personales, lease Amiga de Commodore o el Atari, supieron suplir esta falta desde sus primeros aparatos. El primero, por ejemplo, está dotado de 4 canales de sonido independientes que producen efectos de gran calidad, así como melodias que llegan en muchos casos a ser casi iguales que las originales. Todo un logro, si señor.

Pero el PC era el hermanito feo dentro de la orquesta de los ordenadores. Vamos, que le habían otorgado el papel del que toca la pandereta o el triángulo.

No fue hasta hace poco, que se empezó a luchar contra esto. Compañías como Creative Labs y Adlib lanzaron al mercado las primeras tarjetas de audio con cierta calidad y bajo precio. Ello trajo consigo lo más interesante que puede ocurrir en el ámbito de los ordenadores; la compatibilidad.

Conscientes de ello, los creadores de software, principalmente los lúdicos, supieron sacar partido al invento, acoplando el sonido mediante estas tarjetas a todos sus juegos. Una vez más la pescadilla se mordió la cola. Cuanto más software más tarjetas se vendían y cuantas más tarjetas se vendían, más software se creaba. El resultado del círculo es que el usuario se ve inundado de buen material y calidad sonora en los juegos que adquiere, todo ello gracias a la buena suerte que conlleva el hecho de que algo se estandarize masivamente en un mercado tan susceptible a los cambios.

LOS SONIDOS

Mediante los sonidos logramos tanto o más efectos en el ser humano que las propias imágenes. Si vemos la foto de un niño hambriento en Etiopía, nos lamentamos de este hecho, pero si encima escuchamos sus llantos y sus quejas entonces nuestro corazón sufre todavía más.

Con un sonido logramos alegrar, entristecer o exasperar. ¿quién no ha querido alguna vez darle un par de mamporros a ese automovilista que suena repetidamente su claxon, o a ese motorista que no arregla su tubo se escape y pasa ante nosotros con su ruidos artefacto?, yo creo que a todos nosotros nos ha ocurrido alguna vez.

El sonido es, pues, un lenguaje universal, incluso para los animales. Cuando nuestro perro escucha que el sonido de nuestra voz el alto sabe que algo malo a hecho y que lo regañamos por su comportamiento. Si por el contrario el volumen que utilizamos es dulce y sabrá que no vamos a inferirle ningún daño.

Con los sonidos nos podemos comunicar con los demás, por ejemplo con el Morse y creamos ambientes lugubres y tenebrosos, como en las películas de terror.

Vista su importancia en la vida real y en los juegos de ordenador, pasemos a estudiar qué son en realidad.

LAS PROPIEDADES DEL SONIDO

Un sonido es en realidad el resultado de la vibración de un cuerpo. Se provocan, entonces, unas vibraciones que se expanden en forma de ondas. Estas, mueven el aire, que aun-

que no es un sólido, si está compuesto de partículas como todos los entes terrestres. Gracias a estas partículas que se ven excitadas por la vibración, el sonido se propaga de una parte a otra. Por esta razón, no se transmite el sonido en un lugar que está al vacio, o sea donde no existe ni aire. Las ondas llegan al tímpano y producen la misma vibración de éste, la cual es traducida por el cerebro para que nosotros podamos captarlas.

Esta sería una definición sencilla y comprensible de lo que ocurre cuando damos un zapatazo, cerramos fuertemente una puerta o damos un palmetazo. Normalmente se suele usar el ejemplo del diapasón porque la vibración de éste es tan evidente que incluso puede captarse visualmente.

LA FRECUENCIA

Hemos dicho que el sonido es en realidad una onda. Las ondas pueden variar en "rapidez", es decir en el número de veces que se producen, en ejemplo, por segundo. Así, tendremos ondas más rápidas y más lentas. Este hecho hace que dos sonidos producidos por un mismo objeto puedan sonar de forma diferente. En número de veces que una onda oscila por segundo se denomina ciclo, o Hertzio, proveniente del nombre del físico que estudió el sonido y sus características. El término que describe el número de hertzios de una onda se llama frecuencia. De esta forma tenemos ondas con una frecuencia de 50 hertzios y de hasta 100 kiloherztios, refiriéndose el prefijo Kilo a 1000, como ya habréis adivinado.

La frecuencia es importantísima en lo que a los sonidos se refiere y debemos tener muy claro su significado.

EL VOLUMEN

Ya todos sabéis lo que es el volumen de un sonido, puesto que más de una vez vuestros padres os habrán llamado la atención por lo mismo, cuando ponéis el tocata a toda caña. El volumen, ténicamente, hace referencia a la sonoridad del mismo. Se trata de algo más bien abstracto y difícil de explicar en unas pocas líneas, pero el término sonoridad o intensidad de un sonido puede valernos perfectamente.

EL TONO DEL SONIDO

Otro término que también resulta difícil de describir y que normalmente se describe como el brillo del mismo.

Un sonido puede resultar opaco o brillante aunque por regla general ésto está relacionado directamente con la frecuencia.

Por ahora, tomemos como referencia lo que ya sabemos. Los sonidos pueden ser graves agudos en cuanto al tono se refiere.

En el próximo capítulo repasaremos más aspectos de los sonidos a la par que estudiaremos el listado aparecido en CJ2 sobre los archivos .MOD, que tan relacionados están con los videojuegos comerciales de alta calidad, los cuales, cada vez más, aparecen en el gran mercado del ocio.

No todos somos conscientes de lo importante que el sonido puede llegar a ser en nuestras vidas. Desde el zumbador de nuestro odiado despertador, hasta el dulce chasquido que producen los besos de nuestra pareja todos desempeñan un papel de vital importancia. Y que vamos a decir de nuestros juegos de ordenador. Si nuestra pistola no hace "bang", el coche no hace "Rum, Ruuuum" o el enemigo no grita "Aggg" al morir, parece que no tienen personalidad. Por ello, pretendemos, con este curso, enseñarte a poner la guinda del pastel. Un sonido vale más que mil palabras.



En Octubre de 1994, se pondrá en Marcha el primer BBS privado para programadores de juegos. La inscripción será cerrada, por lo que apuntate cuanto antes. Pide información por correo a la dirección: VIP BBS, C/ Manuel de Falla 26B, 11520 Rota, Cádiz.

La triquiñuela

CODIGO JUEGO

Las teclas

En muchas ocasiones, yo diría que en un noventa y nueve por ciento de los casos, tenemos la necesidad de que el usuario, o sea, la persona que maneja en ese momento el ordenador, introduzca un dato por el teclado o simplemente que conteste a una pregunta. Pero, ¿cómo puede leer el teclado el ordenador?, échale un vistazo a este truco y te enterarás.

Son imprescindibles para que nuestros programas funcionen y sean prácticos. Si el programa no sabe ese dato que tiene que introducir la persona no podrá continuar o simplemente lo que haga no tendrá finalidad ninguna. Y si no que me digan a mí para que sirve un procesador de texto si no podemos introducir el texto. O un programa que pide una clave secreta, si no podemos meterla y arrancarlo.

En definitiva, la comunicación usuario y ordenador debe ser constante y es aquí donde un periférico como el teclado desempeña un papel importantísimo.

Conscientes de ello aquí tienes una rutina echa en lenguaje ensamblador que te permitirá ir conociendo los servicios que el MSDOS ofrece para poderlos utilizar en nuestros programas y sacar beneficio del teclado como medio de comunicación con el ordenador.

Se trata de un instrumento complicado y por ello merecerá toda una sección dentro de Código Juego, pero por ahora te ofrecemos un programa que no hace nada salvo servir de base para estudiarlo.

El programa pregunta si eres Pedro o Juan y en función de la tecla que pulsemos sacará uno u otro mensaje en la pantalla. También, comtempla la posibilidad de que no pulsemos una de las dos teclas de contestación, en este caso J ó P, por lo que no saldrá de un bucle hasta que lo hagamos correctamente.

Se utilizan los servicios del DOS a través de la interrupción 21h y concretamente la parte de estos que tienen que ver con el teclado, motivo de este artículo. En este caso es la función 10h, la cual devuelve en AL el valor ASCII de la tecla pulsada, esperando hasta que esta situación se dé.

Estudiad el listado y haced vuestros propios experimentos. Un saludo.

```
CODIGO      SEGMENT
            ASSUME CS:CODIGO, DS:CODIGO
            ORG 0100H                   ;Programa formato .COM
INICIO      JMP SHORT CORTO

;***********************************************
; LOS DATOS QUE UTILIZAMOS EN EL PROGRAMA
;***********************************************

OK          EQU         13,10           ;retorno de carro y line feed
MENSAJE1    DB          OK,"¿Te llamas (P)edro o (J)uan?","$"
MENSAJE2    DB          OK,"Hola Pedro, llama a tu madre","$"
MENSAJE3    DB          OK,"Hola Juan, tienes que pagar el teléfo-
no","$"

;***********************************************
; PROGRAMA PRINCIPAL
;***********************************************

EMPEZAMOS   PROC NEAR
            MOV AH, 09H
            MOV DX, OFFSET MENSAJE1
            INT 21H


TECLA:      MOV AH,10H
            INT 21H
            AND AL, 0DFH            ;en binario 00001011b
                                    ;se pasa a mayúsculas
                                    ;la tecla pulsada
            CMP AL, "P"             ;tecla P

            JE PEDRO
            CMP AL, "J"             ;tecla J

            JNE EMPEZAMOS           ;no se pulsó una tecla correcta.

            MOV AH,09H
            MOV DX, OFFSET MENSAJE3

            INT 21H
            JMP SHORT FUERA

PEDRO:      MOV AH, 09H
            MOV DX, OFFSET MENSAJE2
            INT 21H

FUERA:      MOV AX,4C00H
            INT 21H

EMPEZAMOS   ENDP
CODIGO      ENDS
            END EMPEZAMOS
```

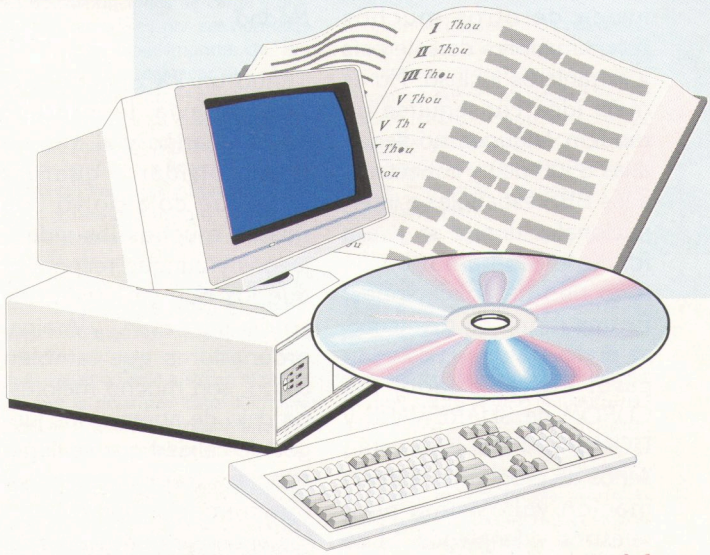
El contestador



Escribe

Joan Esquerra

CODIGO JUEGO



Cada vez interesa más esta sección a los lectores de CJ y así lo prueban la cantidad de cartas que estamos recibiendo. De todas ellas seleccionamos las que nos parecen pueden llegar a despertar más interés por su contenido. Lo que si os prometo es que casi todas llegarán a publicarse, si no ahora, más adelante cuando el nivel de la revista sea mayor.

LA PROGRAMACION DE JUEGOS

Ignacio Vazquez Rovira
Barcelona

Me llamo Ignacio Vazquez y soy un amante de los ordenadores. Puede sonar incongruente que una persona llege a tener amistad con una máquina, pero en mi caso os puedo asegurar que así es. Nuestro amor surje de la mano de muchas horas de mútuo apoyo y comprensión y de largas tardes de compañía cuando nadie quiere estar con nosotros. Ya es mi mejor amigo sin lugar a dudas y gracias a él tengo la diversión asegurada a los mandos de un Ferrari, salvando una guapa heroina en Egipto o pilotando un F16 sobre Siberia. Que amigo te puede ofrecer tanta acción y a continuación ser tan inteligente como para ayudarte con tus estudios del instituto o a llevar la contabilidad de tu negocio.

Pues bien, mi amigo se llama 486DX a 33 Mhz. Tiene un disco duro de 105 megas que corre como el viento y una tarjeta de vídeo Everex. 8 megas de RAM en sus entrañas, ratón y dos disqueras de 3 1/2" de 1.44.
Mi duda es la siguiente:
¿Qué necesito para programar videojuegos?

CJ: Ignacio, tu pregunta es de lo más simple, pero de lo más complicada a la vez. La he escogido puesto que me sirve perfectamente para la explicación de un tema que a todos nos atañe, sobre todo a los lectores de esta revista.
Muy bien, vayamos por partes. Para la generación, programación, desarrollo y creación de un videojuego para ordenador, sea este del tipo que sea, hacen falta 4 cosas fundamentales.
La primera sería el hardware, aunque dentro de esta categoría hay que hacer aclaraciones. Con el ordenador que tu nos dices tener, Ignacio, tienes más que de sobra para empezar a trabajar. Un programador de videojuegos debería tener siempre un equipo lo más completo posible. Por un lado eso nos ayuda a trabajar con velocidad y comodidad. Un gran disco duro, por ejemplo, nos permite tener muchos programas siempre a mano, así como librerías grandes y potentes de gráficos, escenarios, música, sonidos, etc. Por ello, estamos obligados a mejorar nuestra herramienta de trabajo en la medida de lo posible, o en la medida de nuestro bolsillo. Como matiz a resaltar, te diría que de la configuración que nos señalas, se echa en falta una tarjeta de sonidos tipo Soundblaster u otra compatible. Cuando intentes programar la música para tus juegos te la vas a tener que imaginar, por ello es tan necesario equiparnos bien.
La segunda cosa necesaria sería el software. De nuevo me gustaría aclarar algunos puntos: Uno de ellos es distinguir entre los dos tipos principales de programas que nos hacen falta que serían las herramientas de programación y las ayudas a la programación o creación de juegos. Las primeras con los compiladores del lenguaje o lenguajes que deseemos utilizar para el desarrollo de software lúdico, que normalmente serán Ensamblador, C, Pascal, etc. Las ayudas a la programación constituyen todo el resto de programas que nos hacen falta para el diseño de todos los componentes de un juego, como son: programas de tratamiento musical y sonoro (MIDI), de creación y retoque gráfico, programas de animación, programas de edición de texto, y en general todo aquel , por pequeño que sea, que nos sirva para la generación de alguna de las partes de un juego.
La tercera cosa necesaria sería un buen apoyo de material de consulta. Revistas, libros, manuales, etc, incluso una persona o personas que nos ayuden cuando nos quedemos atascados. De sabios es pedir ayuda, por ello, yo

odo lo que

El contestador



mismo, estoy conectado a varios BBS, donde se discuten temas que a todos nos interesan y se pueden encontrar una inestimable ayuda de otros muchos programadores de este país y extranjeros incluso. No es mala idea conseguirse un modem (algunos no llegan a las 8500 pts) y conectarse a los BBS españoles e internacionales donde se vive a fondo la programación de todo tipo de aplicaciones. Es una forma de mantenerse al día sin que cueste tan caro. Por otro lado, los CDROM de SHAREWARE y DOMINIO PUBLICO están llenos de programas e información valiosísima para nuestros videojuegos. Si se sabe buscar, se sabe encontrar.

Y por último, el cuarto requisito sería tener un buen conocimiento del mercado de los juegos para ordenador para saber que tipo de es preferido por los adictos a este tipo de software y conocer si la idea que tu vas a desarrollar ya ha sido creada previamente. Tener una idea original y bien clara sobre lo que queremos hacer se puede decir que es el 25 por ciento del trabajo de la programación de un juego.

Por ello, Ignacio, te recomiendo que sigas nuestras páginas y nuestros cursos, apoyándote en tus amigos informáticos en aquellas cuestiones en las que no tengas las idea muy claras y también, muy importante, mejorando tu ordenador siempre que puedas. En esta revista siempre tendrás y tendréis la ayuda necesaria para llegar a programar vuestros juegos.

ENSAMBLADORES CRUZADOS

Timoteo Roldán Luque
Madrid

Tengo un ZX Spectrum al que todavía le profeso mucho cariño y al que no deseo olvidar o guardar para que coja polvo. Me gustan muchos los videojuegos y aunque reconozco que los que se hacen hoy en día para el PC y Amiga, ordenadores que también tengo, son mucho mejores, no dejo de apreciar los juegos de calidad que se llegaron a hacer para el Spectrum, que durante tantos años fue nuestro barato consuelo informático. Me gusta programar también y aunque no domino ningún lenguaje, me he animado un poco con vuestra revista a la cual considero excelente y amena, aunque un poco complicada para los principiantes negados como yo.

El caso es que estoy seguro de que mi duda o pregunta sí va a interesar a muchos programadores que como yo, están empezando.

He oido y leido por ahí que existen unos programas que te permiten crear programas para otros ordenadores y después probarlos en el ordenador donde los estás creando, ¿es eso cierto?

CJ: Estimado amigo Timoteo, me parece que estás un poquito liado con lo términos que nos usas. Voy a intentar aclarar tus ideas un pelín, pues como muy bien nos dices la pregunta puede ser interesante para muchas personas que programen o estén en el proceso de hacerlo.

El ordenador, como ya muchos sabemos, está controlado por un chip muy importante de su anatomía; el microprocesador. Este, tiene muchas atribuciones y tareas y para realizarlas necesita un programa. El programa está escrito en un lenguaje que el microprocesador puede entender y que es innato a él. Es decir, que el lenguaje que utiliza el procesador del PC no es el mismo que el del Motorola que lleva un Amiga o un Atari, ni tampoco estos dos últimos son comprensibles para la CPU Z80 de un Spectrum.

De ello sacamos la conclusión que un mismo programa tendría que ser escrito en lenguajes diferentes para cada ordenador y que el programa de uno no funciona en el otro.

Pero una cosa es indudable, no es lo mismo programar para PC o para un Spectrum. En el caso del primero tenemos muchos adelantos electrónicos que nos ayudan inmensamente a la hora de crear software (discos duros, microprocesadores, buenos compiladores, ratones, tabletas digitalizadoras, etc.). Sin embargo con el Spectrum todo son problemas al no contar con estos adelantos y facilidad de trabajo. ¿Cuál creéis que es la solución?

Pues la mejor es crear programas para el Spectrum en un PC y después volcar la información en éste último mediante un cable serie.

El principal problema de esto es que el PC no entiende el lenguaje máquina del Spectrum por lo que deberemos utilizar un tipo de programas que se llaman ENSAMBLADORES CRUZADOS. Estos permiten hacer programas en ensamblador para el ordenador final en un ordenador diferente y después compilar el programa y pasar los datos al verdadero ordenador que los va a usar sea un Amiga, Atari, PC, etc.

En el caso del Spectrum existen hasta emuladores. Los emuladores convierten un ordenador en otro, realizando todas las funciones de éste, no debiéndose confundir con el ensamblador cruzado. Con el emulador no tenemos ni que volcar los datos en el Spectrum, tan sólo correr el emulador y probar nuestro programa.

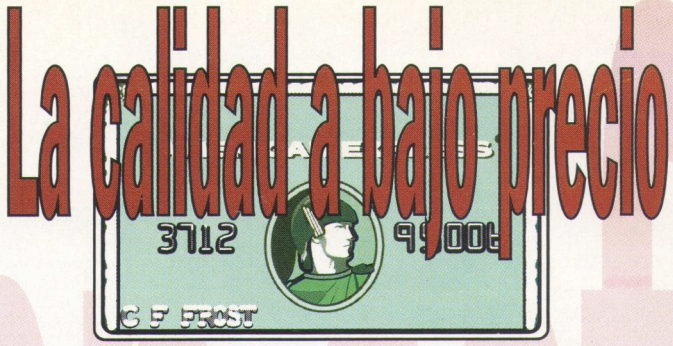
Los ensambladores cruzados son interesantes y se pueden encontrar en muchas de las BBS españolas en los apartados de programación. Para lograr alguno te recomiendo que hables con algúna persona que tenga módem, si es que no lo tienes tú mismo, y que se investigue un poco por algunas de las BBS, ya verás como encuentra lo que andas buscando.

De todas maneras, te comento que en algunos CDROM´s, como el que distribuye SIMTEL se pueden encontrar ensambladores cruzados para varios tipos de microprocesadores.

Esto es todo, espero recibir pronto tu carta con las dudas que quieras que te resuelva.

Shareware

Qedit



La herramienta más usada por el programador, aparte del programa compilador en sí, es el editor de texto con el que se construye el programa línea a línea. Para tal fin se pueden usar prácticamente todos los procesadores de texto que están disponibles.

Desde el Word para Windows hasta el Edit del DOS, pasando por el archiconocido Wordperfect, todos pueden cumplir esta función a la perfección ya que, independientemente de su capacidad para hacer filigranas con el texto, pueden grabar el mismo de forma pura, es decir sin los caracteres de control que cada procesador añade a lo que hemos escrito.

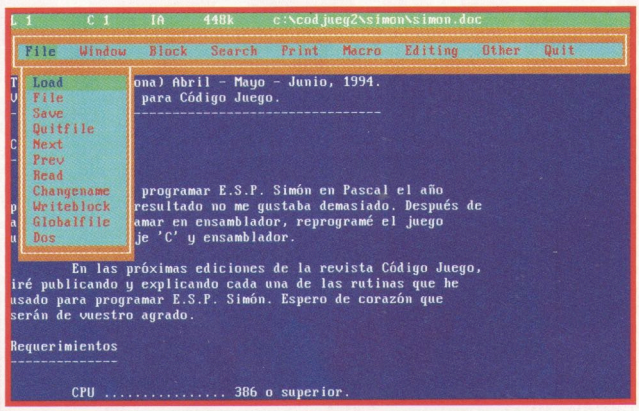
Estos códigos especiales son de uso interno y diferentes en cada procesador de textos, siendo su finalidad la indicar características especiales como el indentado, color de los caracteres, tabuladores, tipo de letra utilizado, formato de negritas, cursivas, subrayado, sombreado, etc. Se puede decir que cuanto más complicado es un programa de este tipo, mayor es el número de caracteres de control que utiliza.

Lo que ocurre es que estos códigos especiales e intrínsecos a cada aplicación de edición de textos, no harían sino confundir al compilador a la hora de realizar su trabajo. Por ello se estableció la norma de que el texto que compone un programa no debería llevar ningún tipo de caracteres de control, en el fondo por que no los necesita realmente. Los programas de ensamblador, C, Pascal, Clipper, por poner un ejemplo, no necesitan en ningún momento más códigos de control que los pocos que permite el sistema ASCII, que es el que se ha establecido como estándar para el tratamiento de la información alfanumérica en el PC y otros ordenadores.

Por ello, encontramos editores de texto, con posibilidad de grabación del archivo en formato ASCII, texto puro o simplemente texto, que debido a su simpleza son una plataforma ideal para trabajar con ellos y finalmente, como es el caso de QEDIT realizar una compilación posterior.

Por supuesto, la falta del editor en los sistemas de programación ha sido subsanada con la aparición de los compiladores contemporáneos los cuales ya incluyen todo lo necesario para realizar un programa de principio a fin. Pero siempre en Código Juego pensamos en aquellos que no tienen un sistema profesional de desarrollo de software y que por tanto todavía siguen trabajando a la antigua. Es decir creando en un editor el programa y realizando una compilación desde la línea de comandos.

Pues bien, el programa que hoy comentamos viene a ayudarnos enormemente en esta tarea de editar ya que ha sido creado como procesador de texto pero con la programación en mente, por lo que encontramos en él varias características y un sistema tan sencillo que lo convierten en una pieza imprescindible de nuestro en nuestro cajón del buen soft.



QEDIT

El programa es shareware es decir que el creador lo ha cedido para su prueba previa y después debemos pagar su uso continuo. Pero ello no es una traba, ya que el registro se paga gustoso, recibiendo a cambio numerosos beneficios. Veámoslos.

En primer lugar, tenemos un editor que nos permite, como otros muchos, establecer los márgenes del texto y tras ello y poco más empezar a trabajar. Los colores de Qedit son totalmente configurables, se pueden, pues, escoger los que más nos gusten, y situarlos en las distintas partes que conforman la pantalla de trabajo de la aplicación, como la barra de información, texto, filo de la ventana, etc.

Qedit tiene 8 menús diferentes que son FILE (archivo), WINDOW (ventana), BLOCK (bloques), SEARCH (buscar), PRINT (imprimir), MACRO, EDITING (edición) y OTHER (otros).

En el primero de ellos encontramos las típicas funciones de carga y grabación, las cuales por ser normales no dejan de tener importancia. Sin embargo, cabe destacar que Qedit permite la inserción de archivos dentro de otros que ya están abiertos y además posibilita trabajar con multitud de archivos a la vez, abriendo diferentes ventanas y saltando entre ellas a nuestro antojo. También encontramos en este menú la opción de abrir una ventana al MSDOS sin dejar la aplicación lo que nos permitirá saltar al sistema operativo siempre que queramos para realizar una operación y después volver a Qedit justo en el punto en que lo dejamos. Esto último es ideal a la hora de compilar algo que acabamos de grabar y si existen fallos de sintaxis o de cualquier otro tipo durante el proceso de compilación, regresar a Qedit, cambiar, editar, grabar y volver a compilar. Así de sencillo.

El menú WINDOW se encarga de ofrecernos las diferentes opciones para trabajar con las ventanas que tengamos abiertas. Podemos aumentar y disminuir el tamaño de las mismas y saltar entre ellas para ir realizando cambios.

El menú de BLOCK contiene las herramientas de marcado y copia de bloques de texto entre ventanas y dentro del mismo archivo.

En SEARCH, se encuentran las opciones de busqueda y cambio de palabras dentro del texto, lo cual todos sabemos lo útil que es a la hora de cambiar el nombre o etiqueta de una variable a lo largo de todo un programa.

Cuando se trata de imprimir todo el listado o sólo parte de él, acudiremos a el menú PRINT que es el que se encarga del trabajo con nuestra impresora.

A continuación hacemos referencia a una de las características más importantes y destacables de este magnífico editor; La posibilidad de utilizar Macros. Seguramente, ya conocéis que es un MACRO, pero para aquellos que no estén familiarizados con ellos, paso a dar una breve descripción.

Durante el trabajo diario con un ordenador, existen muchas, muchísimas tareas repetitivas, las cuales hemos de realizar una y otra vez para desconsuelo y cansancio por nuestra parte. Los ingenieros de software de Microsoft vieron el futuro muy negro a este respecto, así que inteligentemente decidieron crear un tipo de archivo ejecutable por el mismo sistema operativo y que fuese de fácil programación: el archivo .BAT. Gracias a él la vida es mucho más sencilla a la hora de hacer tareas similares una y otra vez.

Pues bien, un MACRO es una especie de archivo .BAT, sólo que dentro de una aplicación. Las reglas de programación de un MACRO son diferentes para cada programa que los crea, pero en definitiva todos tienen la misma estructura y finalidad; evitar que tengamos que hacer las mismas cosas muchas veces.

Suponed que cada vez que termináis de teclear un programa, queréis ir al MSDOS, y ejecutar un archivo .BAT que lo compila y lo ejecuta. La serie de teclas que tendríais que pulsar para realizar esta operación pueden grabarse en un MACRO y después, con una simple combinación de las mismas, todo el proceso se realiza automáticamente, ¿a qué es estupendo?.

Por último deciros que en los restantes menús de Qedit encontramos las opciones de prefencias de usuario para el trabajo con el mismo, como pueden ser la copia de respaldo automática, el autoindentado, rotura automática de palabras, etc.

Nuestra valoración de Qedit no puede ser mejor. Se trata de un programa pequeño y fácil de acomodar a nuestros gustos en cuanto a los colores, teclas, y comportamiento. Los MACROS le dan puerta abierta a los que se esmeren con ellos para realizar infinidad de tareas para convertir nuestro editor de listados en un amigo inseparable.

El próximo mes veremos otra aplicación Shareware para programadores como vosotros.

SIMON

El juego de los colores

Profesor

Eduard Sánchez Palazón

Ya tenemos una sección donde aprendemos a hacer un juego en ensamblador. Ahora, los que quieran aprender a hacerlo en C, ya pueden siguiendo este apartado, donde aprenderás todas las pistas necesarias de la mano de Eduard, nuestro programador. ¡Qué lo

Mediante esta sección podréis saber como se hizo E.S.P. Simón, ya que en este y en los posteriores artículos, iré publicando las rutinas que conforman este juego.

Aparentemente, E.S.P. Simón parece un juego muy sencillo, pero no debemos creerlo así: Simón incluye rutinas y técnicas realmente interesantes que, sin duda, te ayudarán a crear tus propios juegos partiendo, solamente, de tu imaginación y conocimientos.

E.S.P.Simón, como la mayoría de los juegos, se divide en tres partes claramente diferenciadas: la presentación, el nudo y el desenlace. El la presentación, normalmente, se exponen algunas de las rutinas más "espectaculares" del juego, ya que es en este apartado donde el usuario recibe la primera impresión sobre el juego.

La parte correspondiente al núcleo es el juego propiamente dicho, es decir, es donde se desenvuelve la acción, la trama o argumento, y es ahí donde el jugador deberá intervenir para cumplir el objetivo del juego.

Finalmente, el desenlace, corresponde a la parte del juego que une el núcleo con el sistema operativo, es decir, el final del juego.

Una característica importante en un juego es el hecho de que, al cargarlo, éste se "desconecte" totalmente de la tarea que estaba realizando el ordenador en ese momento; por ejemplo, si al cargar el juego, el ordenador estaba generando un sonido para el altavoz, el programa tendrá que encargarse de desactivarlo, por otro lado, si la Sound Blaster estaba tocando una canción deberá resetearse esta tarjeta; de esta forma evitaremos que el "mundo exterior" genere interferencias en nuestro programa.

E.S.P. Simón ha sido programado en lenguaje 'C' utilizando un gran número de rutinas en ensamblador, ya que este lenguaje dispone de una rapidez insuperable en el momento de su ejecución.

Un juego debe programarse de forma muy ordenada: el código fuente debe contener explicaciones de cada una de las funciones, así como de su sintaxis, las variables deben tener nombres coherentes, por ejemplo:

En vez de poner :

```
int y;
```

Debería ponerse:

```
int contador_y;   // Variable usada para el contador 'for'.
```

Estos comentarios y explicaciones nos ayudarán mucho en el momento en que tengamos mil o dos mil líneas de código, ya que así entenderemos la función de cada una de las variables sin tener que rompernos la cabeza.

Otro aspecto muy importante en el momento de programar un juego es que debemos intentar crear nosotros mismos las rutinas según nuestras propias necesidades; al hacerlo, sin duda, aumentaremos nuestro nivel de programación; aunque algunas veces nos vemos obligados a usar las rutinas, programas o ideas creadas por otros autores.

También recomiendo que uséis el lenguaje ensamblador para las operaciones mas concretas (poner pixels, borrar la pantalla, reducir el volumen de la Sound Blaster, etc...). El lenguaje ensamblador permite una gran rapidez de cálculos que, a la larga, influyen mucho en el momento de usar un programa.

Y dados estos sanos consejos, empecemos a analizar el "interior" de E.S.P. Simón un juego que seguro te gustará.

EL JUEGO

En esta primera parte el programa obtiene parte de la configuración del sistema, apaga en altavoz, y enciende las luces del teclado. Vamos a verlo:

El Altavoz

Un aspecto imprescindible para entender este apartado es saber diferenciar el significado de byte, bit, activar un bit y desactivar un bit.

Un Byte es una porción de memoria formada por ocho bits, los cuales los numeramos del siete al cero.

Byte: 7 6 5 4 3 2 1 0 <— Bits.

Los bits, como sabéis, solo pueden tener dos valores: el 1 y el 0. Cuando decimos que se debe activar un bit, nos estamos refiriendo a ponerlo con el valor 1, y cuando lo desactivamos le ponemos el valor 0.

No me extenderé mucho más en ese tema, ya que fue explicado muy claramente en el número anterior de Código Juego.

Lo primero que hace el juego en cargarse es desconectar el altavoz, también llamado PC-Speaker. El PC-Speaker es un instrumento interno del PC que permite generar sonidos a partir de pequeñas vibraciones generadas por el ordenador. El PC-Speaker se controla mediante el puerto 61h. Dentro de este puerto hay un byte (ocho bits) almacenado, del cual solo nos interesan los bits uno y cero.

El resto de bits de este puerto realizan otras funciones que no son de nuestro interés en este momento.

Si queremos desconectar el PC-Speaker solo deberemos coger el byte contenido en el puerto 61h, desactivar los bits uno y cero y volver a almacenarlo en el puerto.

No debemos caer en el error de guardar directamente un valor en este puerto, ya que no sabemos los valores de los otros bits del byte en cuestión. Ver listado 1.

Este pequeño listado en ensamblador permite desconectar el altavoz interno del PC. Vamos a analizarlo detenidamente:

En primer lugar pasamos al registro 'DX' el valor del puerto al que queremos acceder, en este caso el puerto 61h (mov dx,61h). Luego guardamos en el registro de ocho bits 'AL' el contenido del puerto especificado (in al,dx). Mediante la instrucción AND desactivaremos los dos últimos bits del registro 'AL' y mantendremos intacto el resto del byte (and al,11111100b). Ahora el resultado de la operación se encuentra en el registro 'AL', sólo debemos enviar este resultado otra vez al puerto 61h para que el ordenador apague el Speaker.

No sería del todo correcto, aunque posiblemente funcionaría, apagar el PC-Speaker de la siguiente forma, ver listado 2.

El listado anterior también desconectaría el PC-Speaker, pues desactiva los dos últimos bits del puerto, pero de esta forma se modifican los restantes bits y eso, realmente, no es muy recomendable. Es más bien una cuestión de orgullo que no de técnica.

Para conectar el altavoz se debería realizar la operación a la inversa, es decir, activar los bits uno y cero. Para activar dos bits en concreto de un byte se suele utilizar la operación 'OR'. Veámoslo en el listado 3.

La mecánica es exactamente igual a la del listado para desconectar el altavoz, excepto en la penúltima línea: para desconec-

tar los dos últimos bits del puerto usábamos la instrucción AND, mientras que usando la línea ' or al, 00000011b ' conectamos los bits uno y cero, manteniendo intactos los restantes bits (del 7 al 2) del puerto.

El teclado,Lo esencial:

Cuando hablamos de posición de memoria siempre tenemos que dar dos valores: el segmento y el offset. Una posición de memoria siempre se expresa de la siguiente manera:
0000:0417 - El primer valor '0000' corresponde al segmento y el segundo '0417h' corresponde al offset.
Los registros ES y DI del ensamblador son utilizados para expresar el segmento y el offset respectivamente de una posición de memoria.
Para representar una posición de memoria mediante estos registros en ensamblador se escribe ES:[DI], indicando así una posición de memoria basándose en los valores de estos dos registros.
En segundo lugar E.S.P.Simon activa las teclas 'Block num' (o teclado numérico) y 'Scroll Lock' (o bloquear desplazamiento).
Esta operación, aunque no es imprescindible, la da cierta calidad al juego, ya que es el mismo quien activa el teclado numérico.
Para activar esas dos teclas ya no se accede a un puerto, sino a una posición de memoria, concretamente la 0000:0417h. Esta posición de memoria es la que controla las teclas especiales del teclado, tales como bloquear desplazamiento, bloquear las mayúsculas, insert, las teclas shift, etc... Es importante destacar que cada posición de memoria tiene capacidad para un sólo byte.
Si queremos controlar alguna de las teclas especiales, solo deberemos guardar un byte en la posición de memoria 0000:0417h con los siguientes valores de bits, ver listado 4.
Por lo tanto para conectar las teclas 'Bloq Num' y 'Bloqu Despl' solo deberemos realizar lo que se indica en el listado 5.
En la primera línea lo que hacemos es asignar el valor 0 al registro 'AX' y seguidamente poner el valor de 'AX' (que es cero) al registro 'ES', ya que NO podemos poner un valor directamente en el registro 'ES' como 'mov es,0'. Seguidamente ponemos el valor 417h en el registro 'DI'. De esta manera los registros ES y DI contienes los valores 0000 y 417h respectivamente, este valor corresponde a la posición de memoria que queremos modificar. Luego ponemos un valor concreto en el registro 'AL' dependiendo de las teclas que queramos activar o desactivar; en este caso nos basaremos en la tabla de bits anterior.
Como nuestra intención es la de activar las teclas 'Bloq Num' y 'Bloq Despl', activamos únicamente los bits 5 y 4.
Seguidamente guardamos el valor de 'AL' en la posición de memoria indicada por ES:[DI] (0000:0417h) y acto seguido las teclas en cuestión se encenderán.
Otra manera de acceder a la memoria es mediante el propio 'C', usando la instrucción poke y pokeb. Estas dos instrucciones guardan un determinado valor en una posición de memoria determinada. Este valor puede ser de 16 bits, es decir, un WORD, usando la instrucción 'poke', o un BYTE, usando la instrucción 'pokeb'.
Para activar las teclas 'Bloq Num' y 'Bloq Despl' en 'C' se realizaría de la siguiente forma:
pokeb(0x0000, 0x0417, 30);
El número 30 es el equivalente en hexadecimal del número binario 00110000.
A pesar de que usando las propias instrucciones del lenguaje 'C' todo parece más sencillo, recomiendo usar el lenguaje ensamblador para estas funciones tan concretas, ya que, en mi opinión el ensamblador es mucho más lógico.

Detectando la VGA

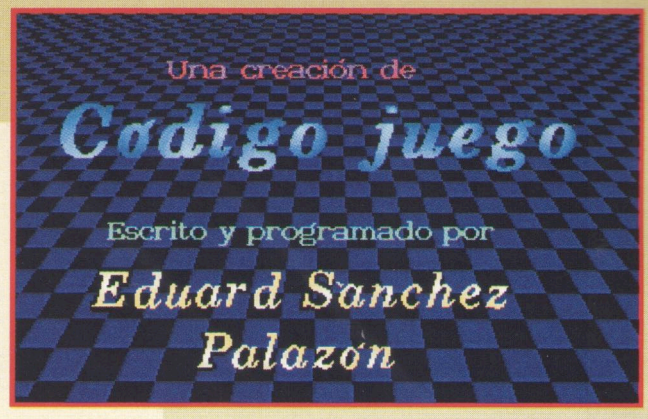
Si has trabajado alguna vez en ensamblador sabrás lo que es una interrupción, además Ramón López os introdujo en el tema mediante el número uno de CJ.
La interrupción es una instrucción en ensamblador que nos permite realizar diversas funciones concretas, como por ejemplo activar el ratón, cambiar el modo de vídeo, poner un píxel en la pantalla, etc...
La interrupción se indica mediante la instrucción 'int <número>'. 'Número' es el numero de la interrupción que queremos ejecutar. Según el valor que le demos a los diversos registros de ocho y dieciséis bits una misma interrupción realizará diversas funciones.
Por ejemplo, para cambiar el modo de vídeo de la tarjeta haremos lo indicado en el listado 6.
Podemos ejecutar una interrupción directamente en ensamblador o bien mediante el propio compilador de 'C'. En este apartado se explicará de las dos maneras.
E.S.P. Simón, al usar gráficos 256 colores, requiere una tarjeta VGA, por lo tanto, antes de empezar a jugar, el programa tiene que comprobar si el ordenador dispone de dicha tarjeta y, si no es así, deberá volver al sistema operativo, ya que resulta imposible la ejecución del programa. De esto se encarga la rutina siguiente, ver listado 7.
La mecánica de esta rutina es muy sencilla: se recurre a una de las muchas funciones de la interrupción de video (10h). Si después de ejecutar esta interrupción con los valores pertinentes, el registro de ocho bits 'AL' no contiene el valor 12h entonces no hay ninguna tarjeta VGA instalada. Entonces se devuelve el valor 0.
Por el contrario, si después de ejecutar la subfunción de la interrupción 10h, el valor de 'AL' es 12h entonces significa que existe una tarjeta VGA instalada, por lo tanto, la rutina devuelve el valor 1.
También se podría detectar la presencia de una VGA sin recurrir al ensamblador, es decir, ejecutando la interrupción 10h desde el mismo lenguaje 'C', ver listado 8.
Otra manera de recurrir a las interrupciones es la siguiente, ver listado 9.

El Ratón

Como muchos otros juegos, E.S.P. Simón requiere que esté instalado un ratón, pero ¿cómo detectarlo?. Afortunadamente existe una interrupción destinada exclusivamente a las funciones del ratón, la 33h.
La interrupción 33h dispone de varias funciones y subfunciones que iremos utilizando a lo largo del programa, pero de momento solo nos interesa la función 0, que es la que nos indica si existe o no un ratón instalado.
Escribiremos, pues, dos sencillas rutinas para detectar la presencia del ratón y para saber de cuantos botones dispone. Si no existe un ratón instalado, el programa saldrá al sistema operativo ya que el ratón, al igual que la VGA, es imprescindible para la ejecución de este programa. Las rutinas son las siguientes, ver listado 10.
Esta primera rutina nos devolverá un 1 ó un 0 dependiendo de si hay un ratón instalado o no.
Debes observar que ya no se usa el ensamblador para llamar a la interrupción 33h, sino que se usa el propio 'C'. Evidentemente se podría haber usado perfectamente el ensamblador, pero es importante ir practicando en acceso a las interrupciones usando otros métodos.
En primer lugar seleccionamos el número de la función, en este caso la 0, poniendo el registro AX a 0. En 'C' para acceder a un registro se hace mediante '_AX = valor'. Luego se ejecuta la interrupción 33h (0x33 para el 'C') mediante la instrucción 'geninterrupt'.
Si después de hacer esto el valor de _AX continua siendo 0 quiere decir que no existe ningún ratón instalado, por lo tanto se devuelve el valor 0. Si el valor de _AX es diferente a 0 el ratón estará instalado y se devolverá el valor 1, ver listado 11.
Una vez detectada la presencia del ratón se ejecutará la misma función de la interrupción 33h. Esta interrupción devuelve en el registro 'BX' el número de botones que tiene el ratón. Este valor se almacena en la variable 'number' y se envía a fuera mediante la instrucción 'return'.
Una vez tenemos la información sobre la configuración del ordenador solo deberemos interpretarla nuestro gusto. Si habéis ejecutado E.S.P. Simón, veréis que aparece en la pantalla un cuadro con unas luces que permanecen encendidas o apagadas según estén disponibles la tarjeta VGA, la tarjeta Sound Blaster y el ratón.
En el número siguiente os mostraré como cargar y manipular imágenes 256 colores, como detectar y resetar la Sound Blaster junto con muchas otras rutinas y parte del código de E.S.P. Simón... Despacito y buena letra!

Eduard Sànchez Palazón
Colaborador de Código Juego.
Junio 1994.





Listado 1:

```
asm {
                mov dx,61h
                in al,dx
                and al,11111100b
                out dx,al
    }
```

Listado 2:

```
asm         {
                mov dx, 61h
                mov al,11111100b
                out dx,al
            }
```

Listado 3:

```
asm {
                mov dx, 61h
                in al, dx
                or al, 00000011b
                out dx,al
            }
```

Listado 4:

Byte que controla las teclas especiales almacenado en 0000:0417h:

```
        Bits          7   6   5   4   3   2   1   0
                      ¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦
 1: Conecta Insert    +   ¦   ¦   ¦   ¦   ¦   ¦   + 1: Conecta Shift Derecha
 0: Desconecta Insert     ¦   ¦   ¦   ¦   ¦   ¦       0: Desconecta Shift Derecha
                          ¦   ¦   ¦   ¦   ¦   ¦
 1: Conecta Bloq Mayus  —+   ¦   ¦   ¦   ¦   +—— 1: Conecta Shift Izquierda
 0: Desconecta Bloq Mayus    ¦   ¦   ¦   ¦           0: Desconecta Shift Izquierda
                             ¦   ¦   ¦   ¦
 1: Conecta Bloq Num     ———+    ¦   ¦   +——— 1: Conecta Ctrl
 0: Desconecta Bloqu Num         ¦   ¦           0: Desconecta Ctrl
                                 ¦   ¦
 1: Conecta Bloq Despl   ————+   +———— 1: Conecta Alt
 0: Desconecta Bloq Despl                        0: Desconecta Alt
```

Listado 5:

```
asm {
                mov ax,0
                mov es,ax
                mov di,0417h
                mov al,00110000b
                mov es:[di], al
            }
```

Listado 6:

```
asm {
                mov ah,0      // Número de la función.
                mov al,13h    // Modo 320x200, 256 colores.
                int 10h
    }
```

Listado 7:

```
int EsVga(void)
        {
        asm {
                mov ah,12h
                xor al,al
                mov bl,36h
                int 10h
                xor ah,ah
                cmp al,12h
                jnz novga
                je vga
            }
        novga:
                return 0;
        vga:
                return 1;
        }
```

Listado 8:

```
int EsVga(void)
        {
                union REGS regs;
                regs.h.ah = 0x12;
                regs.h.al = 0;
                regs.h.bl = 0x36;
                int86(0x10, &regs, &regs);
                regs.h.ah = 0;

                if ( regs.h.al == 12 ) return 1;
                if ( regs.h.al != 12 ) return 0;
        }
```

Listado 9:

```
int EsVga(void)
        {
                _AH = 0x12;
                _AL = 0;
                _BL = 0x36;
                geninterrupt(0x10);
                _AH = 0;

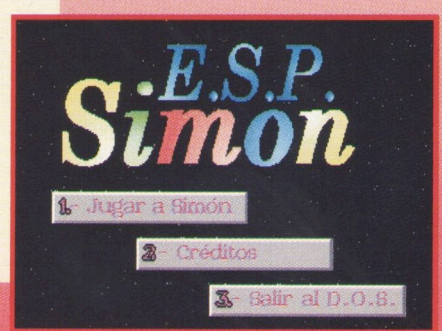
                if ( _AL == 12 ) return 1;
                if ( _AL != 12 ) return 0;
        }
```

Listado 10:

```
int mouse_present( void )
{
        _AX = 0;
        geninterrupt(0x33);
        if (_AX == 0) return 0;
        else return 1;
}
```

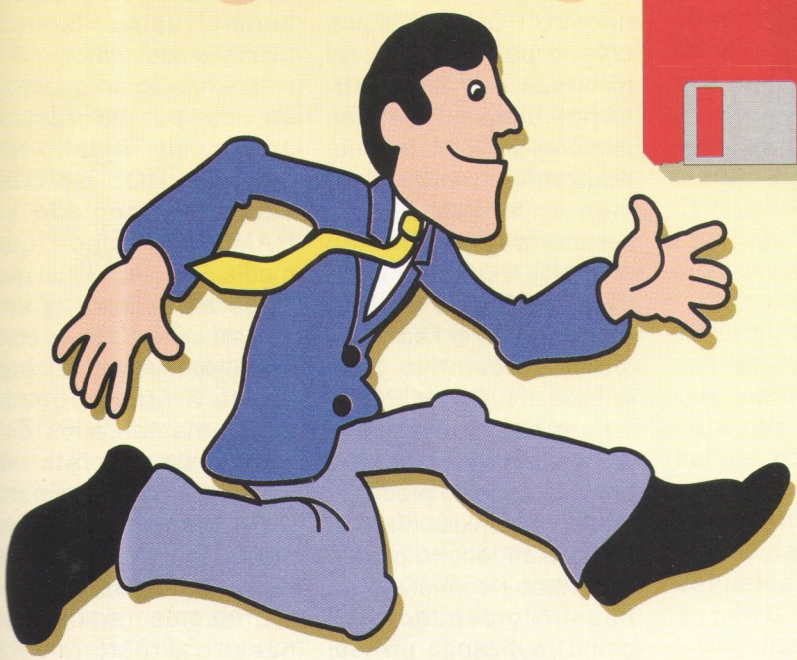
Listado 11:

```
int mouse_botones(void)
{
        unsigned int number;
        asm {
                xor ax,ax
                int 33h
                mov number, bx
            }
        return number;
}
```



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.

CODIGO JUEGO

EL MS DOS

y el programador de videojuegos



Ramón L. Vidal

Como vimos en el número anterior de nuestra revista, el MS-DOS se encuentra guardado en dos archivos diferentes denominados IO.SYS y MSDOS.SYS. Si el sistema operativo de nuestro PC fuese aquel creado por IBM, y no el habitual de Microsoft, dichos archivos tendrían nombres distintos aunque realizarían la misma función en el proceso de arranque. Se llamarían IBMBIO.COM y IBMDOS.COM, pero también conformarían el kernel del sistema operativo y los anexos al BIOS del PC.

Pues bien, vimos que el PC se autocomprobaba con una serie de procesos que en resumidas cuentas analizaban todo lo que se encontraba conectado en nuestro ordenador, así como realizaban un test de su correcto funcionamiento.

Una vez que el ordenador se encuentra en perfecto estado, se procede a la carga del "hermano invisible" o sistema operativo, el cual se responsabiliza del control del PC y de atender nuestras peticiones como buenos usuarios que somos.

Nos habíamos quedado en este punto, precisamente. El registro de carga de DOS o BOOT sector era copiado en RAM (recordad que medía 512 bytes igual que un sector de disco) y una vez allí se le cedía el control, ejecutándolo. El sector de arranque busca inmediatamente los dos archivos esqueleto del DOS y carga en memoria el primero de ellos. Dichos archivos no pueden ser vistos cuando realizamos un DIR de nuestro directorio raiz, sea A: ó C: porque están

dotados de las características de archivos ocultos y del sistema, principalmente para que sean borrados por error. También veo importante deciros que son los dos primeros archivos grabados en el disco y siempre en el mismo orden IO.SYS y después MSDOS.SYS.

IO.SYS

IO.SYS, una vez cargado en memoria, pasa a ser el responsable del siguiente paso del proceso del arranque de nuestro PC. IO.SYS está formado por dos bloques principales. Por un lado las extensiones del BIOS y por otro lado una rutina de arranque que será la que nos interese en este artículo y que se conoce como SYSINIT o inicializadora del sistema. Si queréis una visión más amplia de lo que es el BIOS os recomiendo que leáis el apartado de la VGA de este mismo número de Código Juego. En éste se explica con más detalle que es y para que sirve el otro gran componente software del sistema PC junto con el MSDOS.

SYSINIT

En algunas implementaciones el MSDOS es leido entero en memoria, es decir, tanto IO.SYS como MSDOS.SYS son cargados a la vez, sin embargo en otras, el encargado de leer el segundo archivo es IO.SYS.

Como he dicho, uno de los componentes de este bloque del DOS son los anexos al BIOS que está grabado permanentemente en la ROM del PC. Esta parte contiene los controladores de las unidades definidas por defecto y que siempre están presentes en todos los PC, independientemente de su configuración de periféricos adicionales. Estad unidades son la consola o teclado, el puerto auxiliar (AUX), la impresora (LPT) y los dispositivos de bloques. Además de estos, IO.SYS contiene también pequeños programas especiales que son códigos específicos de hardware que únicamente se ejecutan en el proceso de arranque.

En segundo lugar está SYSINIT, que cuando le toca ejecutarse lo primero que hace es comprobar que cantidad de memoria libre tiene nuestro PC para pasar a autoinstalarse en la parte alta de ésta.

SYSINIT toma el kernel, ó corazón del MSDOS, y lo lleva desde su posición de carga inicial. Entonces elimina todo aquello que perteneciese a IO.SYS y que pueda borrarse porque termino su utilización. SYSINIT se mueve a sí mismo hacia una dirección en la memoria alta donde pueda seguir trabajando, liberando ese espacio para la recolocación de los elementos de MSDOS.SYS. Cuando todo este proceso se ha completado, SYSINIT le da el control a MSDOS.SYS el cual inicializa todas sus tablas de variables en localizadas en la posición de memoria 0000:0400, así como situa las direcciónes de comienzo de las rutinas de servicio de interrupción en la tabla de vectores que se encuentra en 0000:0000. Los vectores que se actualizan van desde 20h al 2Fh que son los que controla el sistema operativo de Microsoft.

En ese momento, el kernel del DOS inicializa todos los componentes de hardware que estén conectados al sistema, mediante una llamada a los drivers (controladores) de cada dispositivo. Con este sistema, el MSDOS permite que un programador de lenguaje ensamblador pueda realizar un programa que conecte un periférico, por raro que sea, al conjunto del sistema operativo, sin que por ello se altere la estructura del mismo.

Los drivers se ocupan de dos cosas fundamentales. En primer lugar comprueban que el elemento de hardware que controlan esté en perfectas condiciones. En segundo lugar, introducen en la tabla de vectores de interrupción los valores de las direcciones de comienzo de las rutinas de servicio que controlan el periferico en cuestión, para que estén allí disponibles para el resto de los programas. El periférico queda así insertado dentro del conjunto global del sistema.

Entonces se procede a determinar de que tipo son las unidades de disco que están conectadas para que el DOS pueda trabajar con el tamaño de sector correcto. Se saca el mensaje del MSDOS por la pantalla y se le devuelve el control a SYSINIT.

Una vez que gracias al fichero MSDOS.SYS se ha podido insertar en las tablas de interrupción los servicios fundamentales del DOS, a través de la interrupción 21h, SYSINIT ya puede leer ficheros, posibilidad que aprovecha para abrir CONFIG.SYS que como sabemos, nos permite configurar el sistema a nuestro gusto.

Son importantes en este archivo los apartados que indican que otros elementos de hardware están conectados al PC (con órdenes como DEVICE=C:\DOS\MOUSE .SYS) así como cuantos BUFFERS, FCBS y STACKS vamos a necesitar. También se indica aquí cual va a ser el procesador de órdenes de usuario que vamos a utilizar. Normalmente será COMMAND.COM, luego se indica con SHELL.

El MSDOS guarda una lista de todos los periféricos que estén conectados así como el inicio de las unidades softaware que los controlan. Cuando CONFIG.SYS se abre se llama a los módulos INIT del driver del nuevo dispositivo, con la orden DEVICE, los cuales indican cuanta memoria necesitan y se pasa, finalmente, a incluir el nuevo periférico en la lista de la que hablamos.

Finalmente, SYSINIT llama a la función EXEC (ejecutar) del MSDOS para correr el programa COMMAND.COM o cualquier otro que indice la orden SHELL de CONFIG.SYS.

COMMAND.COM

Una aclaración me gustaría hacer sobre el inter-

EL MS DOS y el programador de videojuegos

prete de órdenes que sumistra Microsoft por defecto, y es que en realidad está formado por tres bloques diferentes. El primero se conoce como extensión del BIOS. Se trata de una pequeña porción de código que queda unido al MSDOS de forma permanente. El segundo bloque es la que contiene los comandos internos del DOS, como pueden ser COPY, DIR, CLS, TYPE, etc. Este se carga en la parte alta de la memoria convencional de PC para que pueda ser sobreescrito por los programas que necesiten toda la memoria. De esta forma ahorramos memoria ya que los comandos internos de MSDOS no nos sirven para nada cuando estamos dentro de otros programas. Por supuesto, cuando abandonamos el programa y volvemos al DOS, siempre se cargará de nuevo esa parte de COMMAND.COM, en previsión de que hubiese sido borrada por la aplicación.

La tercera y última parte sería aquella que se utiliza tan sólo una vez y después de elimina. Este bloque de COMMAND.COM se encarga de abrir el fichero AUTOEXEC.BAT desde el directorio raiz del disco de arranque. Este fichero es tan importante que nos permite realizar una serie de comandos siempre que arranquemos el ordenador.

Nuestro PC está listo para atender a nuestras peticiones, eso sí sin equivocaciones, porque ya sabéis muy bien lo quisquilloso que es para estas cosas y enseguida nos llena la pantalla de errores.

EL SISTEMA OPERATIVO

Como podéis deducir del proceso de arranque, tanto el POST como los pasos que se siguen en la inicialización del sistema operativo, llevan a cabo una serie de operaciones con una única finalidad en mente; comprobar que el ordenador funciona perfectamente y si no es así no nos deja operar con él.

Esta forma drástica de actuar se debe a que el sistema operativo presupone que toda información que vas a manejar con el ordenador es de gran importancia y no quiere que después de largas horas de trabajo se pierda toda por un error hardware que se detectó en el proceso de arranque. O sea que o el ordenador funciona a las mil maravillas o no hay nada que hacer, pero si lo miráis bien, es mejor así.

Cuando la C: aparece en pantalla, tenemos la seguridad de que todo va bien en las entrañas de nuestro PC.

Pero, ¿cómo funciona el sistema operativo una vez dentro del ordenador?, la respuesta está a continuación.

Originalmente, el MS DOS servía como el agente encargado de una de las operaciones más difíciles dentro de la anatomía del PC, la conexión de varias unidades de disco al sistema. En la actualidad, después de varios años de crecimiento se ha convertido en un lazo de unión total entre nuestro ordenador y el software de aplicación que corre dentro de él.

Sin la ayuda del mismo el PC no sabría enviar información a la impresora, sacar información por la pantalla, interpretar nuestras órdenes, leer y escribir datos en los discos, etc. Entonces, cada persona que comprase un ordenador tendría que escribir un programa que realizase todas estas funciones y muchas más, cosa que derivaría en una incompatibilidad total del software. De hecho, hemos de dar gracias a que tanto el PC de IBM como el MSDOS de Microsoft, se han estandarizado lo suficiente como para que hoy en día dispongamos de buenos precios y de aún mejores y más númerosos programas.

EL MSDOS Y LA MEMORIA

En el proceso de arranque hemos visto una cosa fundamental que deducimos del comportamiento del MSDOS desde un principio: el ahorro de memoria. Incluso si nuestro sistema tiene 16 Megas de RAM, el sistema operativo siempre se comportará de la misma forma, comprimiéndose en la parte baja de la memoria intentando dejar todo el resto para nuestras aplicaciones. Incluso, como vimos antes, parte COMMAND.COM en dos mitades para que la sección del mismo que no se utiliza cuando otra aplicación está funcionando, pueda ser reescrita. Todo un detalle hacia nosotros, los usuarios, que siempre estamos faltos de espacio en la RAM.

Una aclaración: La memoria del PC va desde la posición 0 a la posición 655360 que son exactamente 640 k, que es la llamada memoria convencional o base, y aunque hay más memoria tras ésta, por supuesto, la que nos interesa es la primera que hemos mencionado.

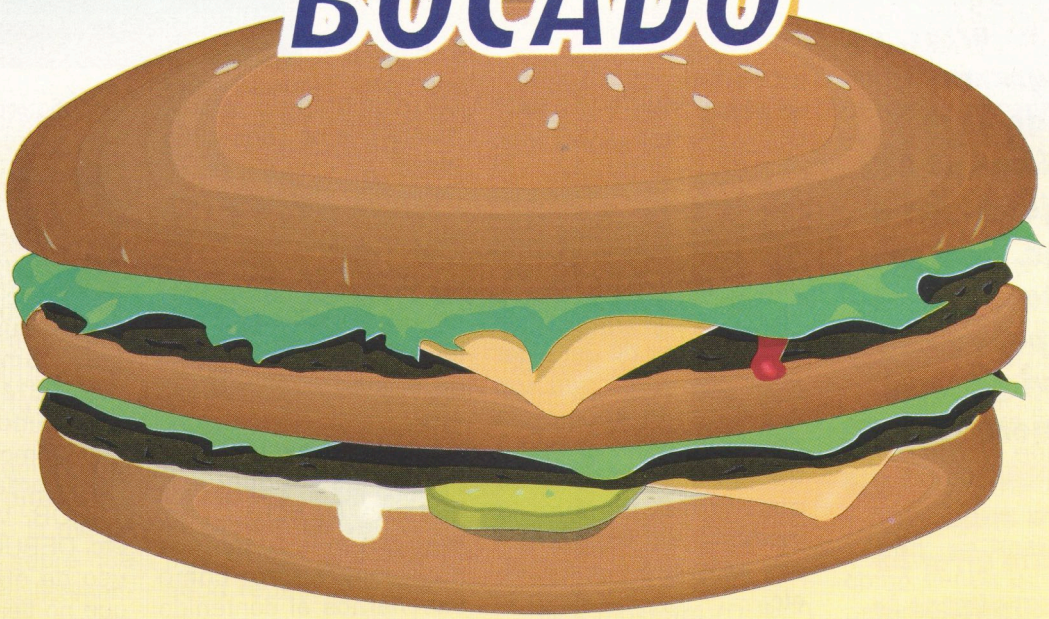
Sólo disponemos de 640 k para que nuestros programas puedan correr libremente y el MSDOS ha de estar dentro de esta memoria, por lo tanto cuanto más pequeño sea mejor. Y hacia esto ha tendido el sistema operativo de Microsoft desde un principio, hacia lo compacto, pero a la vez potente. La convivencia del DOS con nuestros programas ha de basarse en una buena amistad.

En el próximo número estudiaremos como el MSDOS reparte sus componentes en la memoria convencional de nuestro PC.

SUBSCRIBIRSE A

CODIGO JUEGO

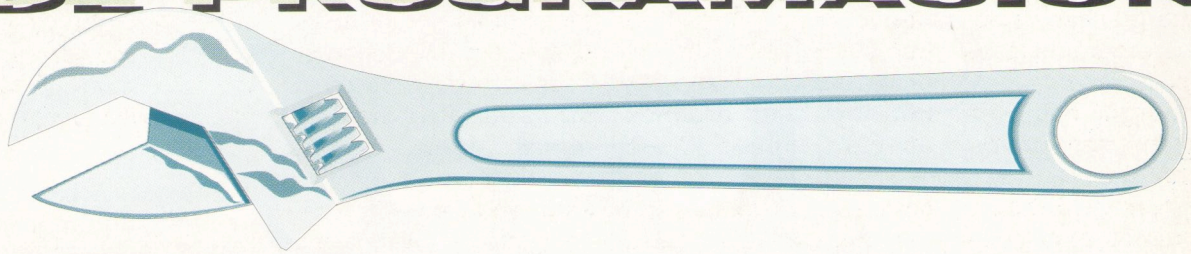
ES UN BUEN BOCADO



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



LOS DEBUGGERS

Colaborador

Barón Rojo Bytes

¡AYi, Tengo un error, en mi programa. ¿Qué puedo hacer? Tal vez si llamo al 092 , no..., mejor llamo a Peter Norton, para que venga a echarle un vistazo, o le pongo una vela a la virgen del Carmen.

Seamos realistas, nada de esto vale para nada cuando ese maldito error ha venido a amargarnos la existencia.

Tan sólo necesitamos una buena herramienta de depuración y mucha paciencia.

DEBUG no es un gran depurador, pero nos ayudará a aprender como usarlos.

Seguimos, este mes, complementando al curso de lenguaje ensamblador que también de publica en Código Juego. El uso de DEBUG, se nos antoja imprescindible a la hora de aprender unos conceptos que son básicos para el correcto entendimiento de como trabaja nuestro PC y de que forma maneja la memoria, las interrupciones hardware y software, etc.

Como hemos visto hasta ahora, un depurador de código o debugger en inglés tiene como principal misión desensamblar nuestro programa en código máquina y monitorizar su funcionamiento paso a paso. Gracias a ello, encontramos los errores que impiden que el programa realize su función correctamente o simplemente que no llege a funcionar de ninguna de las maneras. Debido a esta particularidad de permitirnos avanzar despacio por las entrañas del código, los depuradores son para los programadores como los bisturís para los cirujanos.

Aparte de esta función, todos los debuggers, incluido DEBUG.COM, realizan más funciones suplementarias como enseñarnos el contenido de los registros en todo momento, mostrarnos la memoria, grabar el programa cambiado, etc.

El pasado mes comenzamos a estudiar cómo DEBUG puede ensamblar nuestro programa directamente, eso sí, con restricciones. Una de éstas, yo diría que la más importante, es la de no poder utilizar etiquetas que nos ayudan a referirnos a las variables y las posiciones de memoria con nombres asignados por nosotros. Así, en lugar de recordar una posición de memoria por un número hexadecimal como pueda ser 0003:2314 simplemente le asignamos un nombre que siempre es más fácil de recordar, como CONTADOR, VIDAS, BALAS, etc. DEBUG no autoriza el uso de etiquetas puesto que no tiene un interprete de las mismas, realizando lo que se conoce como ensamblado en un sólo paso.

Precisamente fue en este lugar donde nos quedamos en el anterior número de Código Juego. Aprendimos como, con la orden A(ssemble), el depurador entraba en modo de ensamblado.

Entonces, nos aparece un cursor en la pantalla invitándonos a introducir las sentencias de nuestro programa.

Como ya habréis pensado seguramente, DEBUG no ofrece grandes ventajas a los programadores de lenguaje ensamblador, sin embargo, como he comentado anteriormente, nos permite hacer nuestros primeros pinitos y experimentos para comprobar la manera en que el lenguaje ensamblador se transforma en lenguaje máquina comprensible por nuestro microprocesador. Por ello vamos a continuar con el estudio de los comandos de DEBUG.

Pero antes, un alto en el camino para aprender algunas convenciones que vamos a utilizar a partir de ahora. La sintaxis de DEBUG es muy rígida, con lo que el más mínimo error nos hará dar de bruces con mensajes odiosos sobre los fallos cometidos en la introducción de órdenes. Es, pues necesario, aclarar un poco, llegado este punto, como funciona la sintaxis de todas las órdenes de DEBUG.

Para cada comando la sintaxis es diferente, dependiendo de la misión que realizen y sobre que lo realizen específicamente, pero todas reunen una características comunes.

```
ORDEN [argumentos]
```

Los argumentos pueden ser o no ser necesarios así que los ponemos entre corchetes para explicar esta condición.

La orden es simpre de una única letra que abrevia el nombre de un comando. Así, tenemos A de Assemble, W de Write (escribir), R de Registro, U de Unassemble (desensamblar), etc.

Tras ésta y sin dejar un hueco entre ellos si no queremos, aparecen los argumentos o parámetros que todas las órdenes aceptan menos Q de QUIT (abandonar, salir del programa). Estos pueden ser, dependiendo de la sentencia, vuelvo a repetir;

1) UNIDAD:, un número hexadecimal de un sólo dígito que indica de que unidad lógica de disco, de las que están conectadas a nuestro sistema, es la elegida para la carga o grabación de datos 0=unidad A, 1=unidad B, 2:unidad C, etc.

2) BYTE : Un digito de 2 cifras hexadecimales que se coloca en un registro o posición de memoria.

3) RECORD: Un valor de 1 a 3 cifras hexadecimales que indica el lugar lógico de dentro de un disco así como el número de sectores que se van a cargar o grabar.

4) UN VALOR: Un valor de hasta 4 dígitos hexadecimales que especifican un número de un puerto o el número de veces que una orden o comando debe repetir sus funciones.

5) DIRECCION: Un parámetro que tiene dos partes separadas por dos puntos (:). La primera consiste en un número hexadecimal de 4 cifras que indica el segmento de la dirección de memoria. También, en caso de que hagamos referencia a los segmentos que el microprocesador utiliza por defecto (CS, DS, ES, SS), se pueden utilizar los nombres correspondientes. La segunda parte de la dirección es un número de 4 cifras hexadecimales que indican el OFFSET o desplazamiento dentro del segmento indicado por la primera cifra. (nota: la manera en que la CPU trabaja con la memoria segmentada es materia del curso de ensamblador y se verá en su momento). También, para indicar el desplazamiento se puede indicar el registro IP o puntero de programa. Así, algunas ordenes válidas con el parámetro DIRECCION serían:

```
A 0001:1426
A CS:2345
A CS:IP
```

6) RANGO: Indica un principio y un fin dentro de la memoria del PC, o sea un rango de posiciones de memoria sobre el que queremos actuar. Puede ser dos parámetros DIRECCION, que se explicaron en el punto número 5, indicando la primera el inicio y la segunda el final. Por ejemplo:

```
D 1234:0890 1234:3456
D CS:1010 1100
```

Nos mostrará (DUMP) el contenido de las posiciones de memoria contenidas entre esas dos direcciones de memoria.

También, se puede especificar un rango mediante el parámetro L que significa LONGUITUD:

```
D 1234:0000 L 8
```

Mostrará los ocho bytes que hay tras la dirección 1234:0000

LISTA: Una cadena de números hexadecimales como los que se usan cuando queremos encontrar una cierta combinación de bytes dentro de la memoria.

```
F CS:100 4B 42 5D
```

La orden anterior buscará los números 4B 42 5D en el segmento de código a partir de la dirección 0100 hexadecimal.

6) CADENA O STRING: Un número cualquiera de caracteres entrecomillados con comillas dobles (“”) o simples (‘ ‘). Los valores ASCII de esos caracteres son los que sirven en realidad, para ello DEBUG realiza la traducción automáticamente.

```
F DS:0 “Búscame”
```

La orden anterior intentará encontrar en el segmento de datos la cadena “Búscame”. No os preocupéis sólo la uso como ejemplo ya que la orden F de FIND, o encontrar, todavía no la hemos visto.

Bien, ya hemos visto como tienen que ser los parámetros que usemos como argumento de las órdenes del depurador DEBUG.

El mes que viene continuaremos viendo ejemplos de la utilización de estos comandos, así como empezaremos a desensamblar y estudiar un programa paso a paso.

Listado DEMO.PAS
Gráficos 3D

Listado complementario del curso de Pascal en página 20

Este programa se incluye en el super-disco número 3

```
program p3d;

uses crt,
     graph;

const
     precision = 0.01;
     delaytime = 0;          (* ajustala segun tu ordenador *)

type attr = record
          rx : real;                    { Angulo de giro sobre eje X }
          ry : real;                    { Angulo de giro sobre eje Y }
          rz : real;                    { Angulo de giro sobre eje Z }
          tx : real;                  { traslacion en eje X }
          ty : real;                  { traslacion en eje Y }
          tz : real;                  { traslacion en eje Z }
          d  : real;                  { distancia focal }
end;

type coor = record
     xx:integer;
     yy:integer;
end;

type cuad = array [1..10,1..10] of coor;

var x,y,z,e,f:real;
    aa,a,b,c,g,grmode,grdriver,sx,sy:integer;
    at:attr;
    op:char;
    sw:word;
    co:cuad;
    costbl,sentbl:array [1..630] of real;
    ii:real;


(* saca el seno de la tabla de senos *)
function seno(s:real):real;
begin
     seno:=sentbl[round(s/precision) mod 629];
end;


(* saca el coseno de la tabla de cosenos *)
function coseno(s:real):real;
begin
     coseno:=costbl[round(s/precision) mod 629];
end;



(* procedumiento principal que hace la proyeccion a 3D *)
procedure pasa (x,y,z:real ; atributo:attr ; var sx,sy:integer);
var bx,by,bz,xa,za,ya,cr1,cr2,cr3,sr1,sr2,sr3:real;


begin
     sr1:=seno(atributo.rx);  cr1:=coseno(atributo.rx);          (* calculo de senos y cosenos *)
     sr2:=seno(atributo.ry);  cr2:=coseno(atributo.ry);          (* para el giro de DERIVA,    *)
     sr3:=seno(atributo.rz);  cr3:=coseno(atributo.rz);          (* de BALANZA y de CAIDA.     *)

     x:=-x;
     xa:=cr1*x-sr1*z; za:=sr1*x+cr1*z;                          (* rotacion de deriva         *)
     x:=cr2*xa+sr2*y; ya:=cr2*y+sr2*xa;                         (* rotacion de balanza        *)
     z:=cr2*za-sr3*ya; y:=sr3*za+cr3*ya;                        (* rotacion de caida          *)
     x:=x+atributo.tx; y:=y+atributo.ty; z:=z+atributo.tz;      (* traslacion                 *)
     sx:=trunc(atributo.d*x/z); sy:=trunc(atributo.d*y/z);      (* proyeccion                 *)
end;


(* dibuja un cuadrado en pantalla en 3D *)
procedure cuadro(at:attr ; var co:cuad ; b:integer);
begin
     pasa(30,-30,30,at,sx,sy);
     co[1,b].xx:=sx;
     co[1,b].yy:=sy;
     moveto(sx,sy);
     pasa(30,-30,-30,at,sx,sy);
     co[2,b].xx:=sx;
     co[2,b].yy:=sy;
     lineto(sx,sy);
     pasa(-30,-30,-30,at,sx,sy);
     co[3,b].xx:=sx;
     co[3,b].yy:=sy;
     lineto(sx,sy);
     pasa(-30,-30,30,at,sx,sy);
     co[4,b].xx:=sx;
     co[4,b].yy:=sy;
     lineto(sx,sy);
     pasa(30,-30,30,at,sx,sy);
     co[5,b].xx:=sx;
     co[5,b].yy:=sy;
     lineto(sx,sy);
end;


begin

(* genera la tabla de senos y cosenos *)
     aa:=0;
     ii:=0;
     repeat
          costbl[aa]:=cos(ii);
          sentbl[aa]:=sin(ii);
          inc(aa);
          ii:=ii+precision;
     until ii>=PI*2;



     grdriver:=detect;                          (* activa modo grafico *)
     initgraph (grdriver,grmode,'');

(* atributos iniciales de los puntos *)
     at.d:=1200; at.rx:=5.68319; at.ry:=6.28319; at.rz:=5.79778;
     at.tx:=-100; at.ty:=-70; at.tz:=-350;


(* EFECTO 1 *)
(*=========================================================*)

   at.tz:=-2000;               (* ajusta posicion Z inicial *)
   b:=1; c:=1;
   repeat
     setcolor(9);

     cuadro(at,co,b);          (* dibuja cuadrado *)

     at.rx:=at.rx+0.2;         (* actualiza angulo de giro *)
     at.tz:=at.tz+20;          (* actualiza posicion Z *)
     delay(10);

     setcolor(0);
     moveto(co[1,c].xx,co[1,c].yy);             (* borra las caras *)
     for a:=2 to 5 do
          lineto(co[a,c].xx,co[a,c].yy);

     delay(delaytime);
   until (at.tz>-350) or (keypressed);


(* EFECTO 2 *)
(*=========================================================*)



   b:=0; c:=1; e:=0.01;
   repeat
     if b=10 then b:=0;
     if c=10 then c:=0;
     setcolor(9);

     cuadro(at,co,b);

     at.rx:=at.rx+e;
     e:=e+0.0004;

     setcolor(0);
     moveto(co[1,c].xx,co[1,c].yy);
     for a:=2 to 5 do
          lineto(co[a,c].xx,co[a,c].yy);

     inc(b);
     inc(c);
     delay(delaytime);

   until (e>0.15) or (keypressed);

(* EFECTO 3 *)
(*=========================================================*)

   e:=0.01;
   repeat
     if b=10 then b:=0;
     if c=10 then c:=0;
     setcolor(9);

     cuadro(at,co,b);

     at.rx:=at.rx+e;
     at.ry:=at.ry+e;
     at.rz:=at.rz+e;
     e:=e+0.0001;

     setcolor(0);
     moveto(co[1,c].xx,co[1,c].yy);
     for a:=2 to 5 do
          lineto(co[a,c].xx,co[a,c].yy);

     inc(b);
     inc(c);
     delay(delaytime);

   until (e>0.05) or (keypressed);


(* EFECTO 4 *)
(*=========================================================*)



   repeat
     if b=10 then b:=0;
     if c=10 then c:=0;

     setcolor(b);

     cuadro(at,co,b);

     at.rx:=at.rx+e;
     at.ry:=at.ry+e;
     at.rz:=at.rz+e;
     e:=e-0.0001;

     setcolor(0);
     moveto(co[1,c].xx,co[1,c].yy);
     for a:=2 to 5 do
          lineto(co[a,c].xx,co[a,c].yy);

     inc(b);
     inc(c);
     delay(delaytime)

   until (e<0.005) or (keypressed);


(* EFECTO 5 *)
(*=========================================================*)

   cleardevice;
   f:=-7; b:=1; c:=1; g:=1;
   repeat
     setcolor(9);

     cuadro(at,co,b);

     at.rx:=at.rx+0.02;
     at.ry:=at.ry+0.02;
     at.rz:=at.rz+0.02;
     at.tz:=at.tz+f;

     delay(10);

     setcolor(0);
     moveto(co[1,c].xx,co[1,c].yy);
     for a:=2 to 5 do
          lineto(co[a,c].xx,co[a,c].yy);


     if at.tz<-1000 then
     begin
          f:=10;
          inc(g);
     end;
     if at.tz>-350 then f:=-30;
     delay(delaytime);

   until (g>6) or (keypressed);



(* EFECTO 6 *)
(*=========================================================*)

   f:=-7; b:=0; c:=1;
   repeat
     if b=10 then b:=0;
     if c=10 then c:=0;
     setcolor(9);

     cuadro(at,co,b);

     at.rx:=at.rx+0.02;
     at.ry:=at.ry+0.02;
     at.rz:=at.rz+0.02;
     at.tz:=at.tz+f;

     setcolor(0);
     moveto(co[1,c].xx,co[1,c].yy);
     for a:=2 to 5 do
          lineto(co[a,c].xx,co[a,c].yy);

     inc(b);
     inc(c);

     if at.tz<-1000 then f:=7;
     if at.tz>-350 then f:=-30;
     delay(delaytime);

   until keypressed;
   closegraph;
end.
```

En el número de septiembre

Novedades

La música llega a MOOKIE

Añadiremos a nuestro juego las melodias que acompañarán a la presentación.
Aprenderás como introducir música en tus propios juegos. No te lo pierdas.



Curso El AMOS de AMIGA

Además del curso de ensamblador iniciado este mes, los usuarios de Amiga están de suerte con este nuevo curso sobre un lenguaje de programación que permite hacer maravillas.

El juego del mes

Cool Spot



Ajatel S.L.

Ajatel S.L.

Este verano habrá Mucho Sol



y hará calor

Si no quieres aburrirte como una ostra, compra la única revista que de verdad te enseña a programar

CODIGO JUEGO

Código Juego es una revista de enseñanza de programación. Contiene cursos de lenguaje C, ensamblador PC, Amiga y Atari, Pascal, MIDI, sonidos, animaciones, trucos, entrevistas, preguntas y respuestas, etc... No lo dudes es la única en su género en todo el mundo