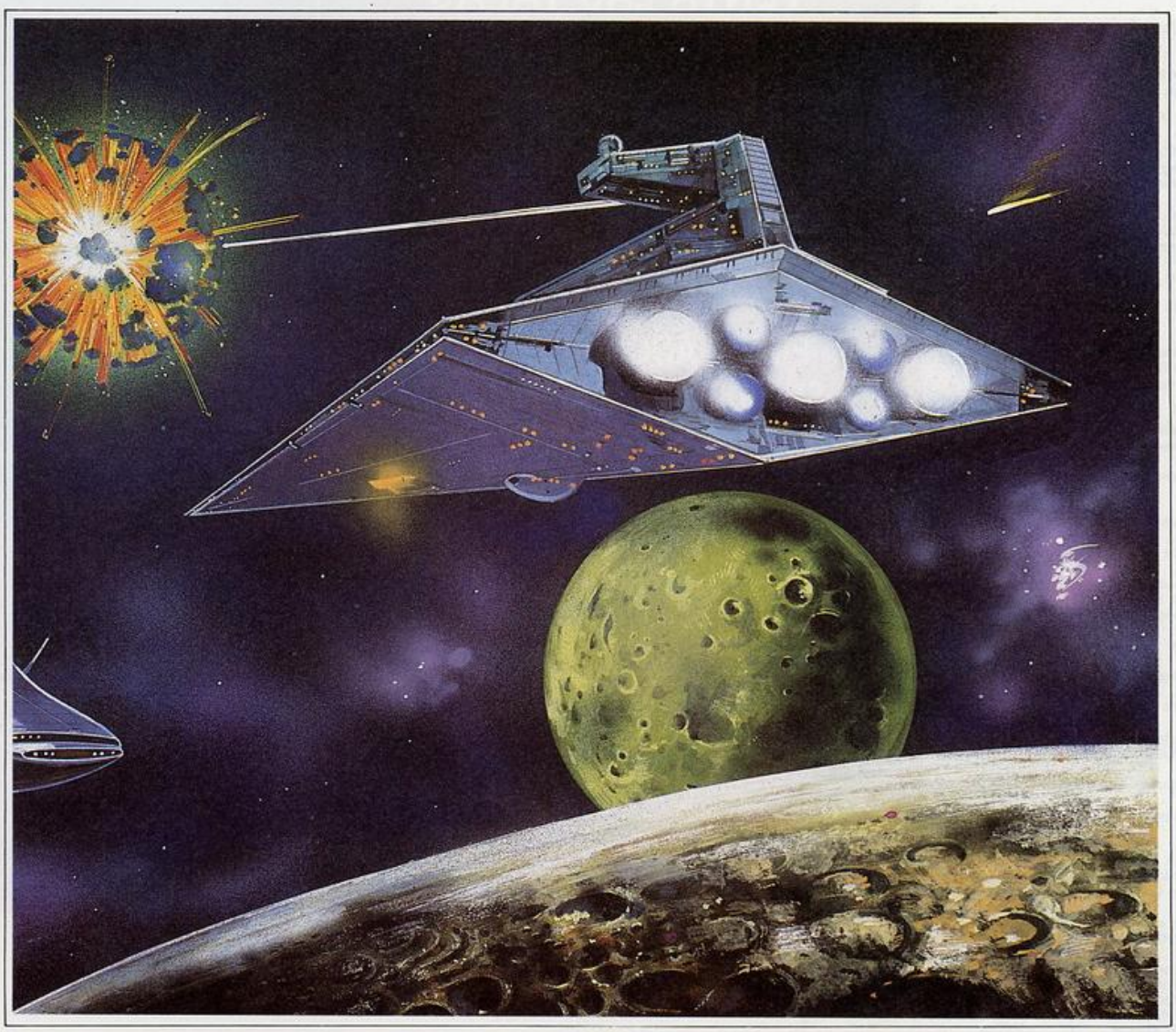
35
150 pts.

RUN

Enciclopedia Práctica del Spectrum



Nueva Lente/Ingelek

EMPLEO DEL CODIGO MAQUINA

EL denominado código máquina es el lenguaje de más bajo nivel entre los existentes, es decir, aquel que requiere un mayor esfuerzo de programación. Sin embargo, sus prestaciones son grandes, por ser también el que menor cantidad de memoria necesita y a la vez, e incomparablemente, más rápido que cualquier otro.

Por todo ello, el código máquina es el lenguaje comúnmente utilizado en los programas comerciales, los cuales por su volumen de ventas permiten amortizar el mayor costo de programación que representa el realizar programas de este tipo.

El código máquina es el único lenguaje que «comprende» el microprocesador Z80A de nuestro Spectrum, así como todos los ordenadores «comprenden» el correspondiente al microprocesador contenido en su unidad central de proceso (CPU).

Es evidente, sin embargo, que el empleo del código máquina conlleva dos importantes problemas: la dificultad de programación y la imposibilidad de ejecutar estos programas en otro ordenador diferente de aquel para el cual fueron concebidos.

El código máquina es el único lenguaje comprendido directamente por un microprocesador.

EL LENGUAJE MAQUINA

El código máquina es un lenguaje de bajo nivel, pero son muchas sus prestaciones.

El lenguaje máquina o código máquina está constituido por una serie de instrucciones inteligibles directamente por el microprocesador, las cuales ejecutan acciones muy concretas y simples. No obstante, esta simplicidad aparente de las instrucciones se ve potenciada al máximo debido al gran número de las disponibles en el vocabulario del lenguaje.

De hecho, la mayoría de los microprocesadores llegan a manejar más de 200 instrucciones, mientras que el vocabulario BASIC se reduce, por lo general, a no más de 80 sentencias. Además, debemos tener en cuenta que con el uso exclusivo de unas 20 de ellas, es posible codificar casi cualquier programa, por existir gran cantidad de las mismas que de forma específica se emplean en determinada área: logaritmos, trigonometría, control del sonido, color, alta resolución de pantallas, etc...

¡!

No existe ningún lenguaje ensamblador incorporado en la ROM de nuestro Spectrum, por cual si deseamos codificar algún programa en este lenguaje, debemos recurrir a programas ensambladores de firmas comerciales presentes en el mercado.

*

Los códigos de operación y los datos, son distinguidos por el microprocesador según el orden en el cual se suministran, dado que externamente no presentan ninguna diferencia: ambos son bytes.

La programación en código máquina exige un esfuerzo superior al realizado en BASIC.

Por otra parte, al igual que las instrucciones BASIC se pueden componer de palabras clave más complementos (parámetros, operandos, funciones, etc...), el código máquina emplea códigos de operación simples y compuestos; donde estos últimos se ven acompañados de determinados complementos u operandos, denominados generalmente DATOS.

En cuanto al modo de interpretar las instrucciones, debemos saber que el microprocesador no distingue un código de instrucción de un dato, más que por el orden en que éstos le son suministrados, puesto que todos le llegan como bytes de 8 bits.

De esta forma, un programa en código máquina no es más que una secuencia de bytes más o menos larga, que sólo adquieren verdadero sentido al ser interpretados por el microprocesador, el cual, por su disposición, sabrá descifrar cuáles son las instrucciones y dónde se encuentran los datos.

La zona de los UDGs está permanentemente protegida, incluso contra un NEW.

EL LENGUAJE ENSAMBLADOR

En un intento de aproximar el lenguaje máquina a uno de alto nivel, el cual permitiera la más fácil codificación y comprensión de los programas, se comenzaron a diseñar los lenguajes ensambladores.

El lenguaje Ensamblador *(Assembler)* tiene muchas ventajas con respecto al código máquina. La más importante de ellas, desde luego, es que el programador no tiene que recordar las instrucciones del lenguaje por su cógido numérico, pues permite el empleo de nombres simbólicos (mnemónicos) mucho más fáciles de memorizar.

Así, por ejemplo, para recordar el código de operación que realiza la siguiente misión: cargar (almacenar en) el acumulador (A) con el contenido del siguiente byte (el siguiente byte será por tanto un dato, por ejemplo 10), no cabe duda que será más fácil utilizar el mnemónico ensamblador **LD A,10,** que el correspondiente código máquina: 62, 10.

Sobre todo, teniendo en cuenta dos factores: en primer lugar, los mnemónicos tienen el significado de abreviaturas inglesas (**LD**=*LOAD*=CARGAR), y en segundo lugar, que si sólo existiese un código de operación, como por ejemplo el 62, sería una cifra tan fácil de memorizar como la palabra LD, pero en el caso concreto del Z80, se superan con creces los 500 códigos de operación con misiones diferentes.

Asimismo, el ensamblador permite que el listado de un programa resulte descifrable directamente por nosotros sin ayuda de la tabla de códigos de operación, proporcionando un sin fin de posibilidades, como la de emplear «etiquetas» en las bi-

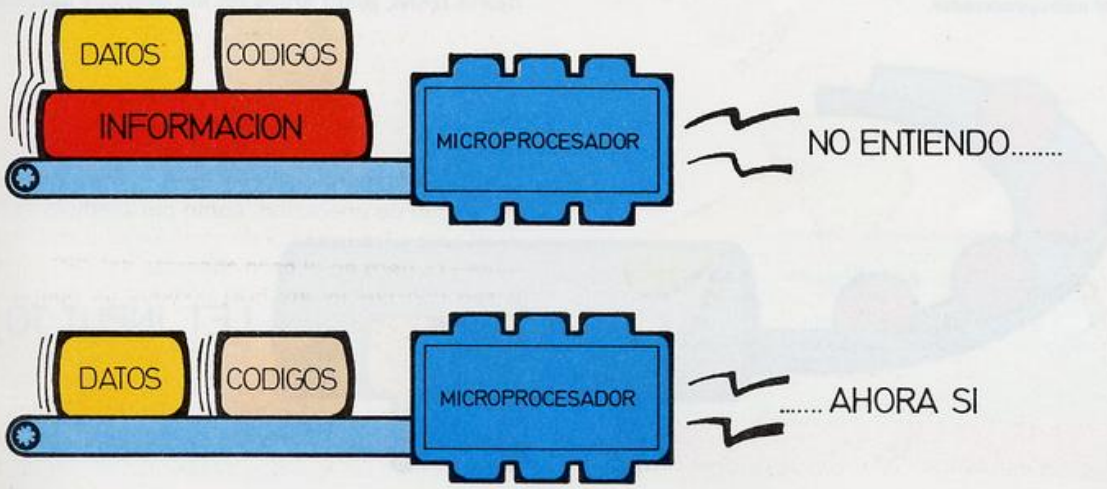
El vocabulario manejado por el código máquina es de considerable extensión.

furcaciones del programa (el equivalente BASIC de **GO TO** y **GO SUB** a un número de sentencia), así como nombres simbólicos para las variables, similares a los que podemos emplear en BASIC.
Además, el Ensamblador se encarga de efectuar la traducción de lo codificado al código máquina estrictamente hablando, para ubicarlo en el lugar de la memoria que le indiquemos.
No existe ningún Ensamblador incorporado en la ROM del Spectrum, como sucede con el intérprete BASIC. Por ello, si nuestro deseo es codificar rutinas en este lenguaje para optimizar nuestros programas BASIC, o para crear otros nuevos directamente, debemos hacernos con uno de los muchos elaborados por conocidas firmas comerciales.

UBICACION EN LA MEMORIA

Si desgraciadamente no disponemos de un Ensamblador, deberemos codificar nuestras subrutinas directamente en código máquina con ayuda de la tabla de códigos de las diferentes instrucciones, para colocar luego la serie de bytes obtenida en un lugar protegido de la memoria donde podamos llamar a la subrutina desde el BASIC.
La mejor forma de proteger un programa en código máquina es colocarlo muy por encima del texto BASIC y su área de variables, con objeto de evitar el posible solapamiento, debido al crecimiento incontrolado de la zona BASIC, y poniendo cuidado en rebajar convenientemente el RAMTOP (Variable del Sistema que indica el tope de la memoria empleable por el BASIC) para que la protección sea realmente eficaz.
Si recordamos lo que se dijo sobre la estructura general de la memoria del Spectrum, nos será fácil situar estos programas, sabiendo que la posición más alta de la memoria en el modelo de 16 K corresponde a la 32767 y en el de 48 K a la 65535.

El microprocesador no distingue un código de operación de un dato, más que por el orden en el cual son suministrados.

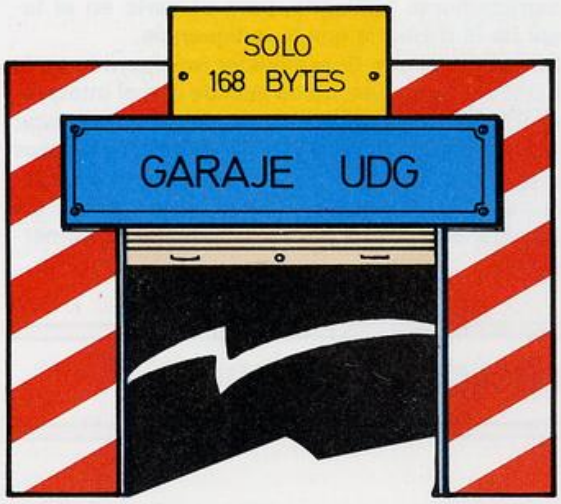


¡!

La mayoría de los microprocesadores llegan a manejar más de 200 códigos de instrucción diferentes, mientras que el vocabulario BASIC no supera generalmente las ochenta.

*

Cualquier problema en el manejo del código máquina, por grave que parezca, se puede solucionar apagando y volviendo a encender el aparato.



Podemos almacenar bytes en la zona de los UDGs en un número no superior a 168.

¡!

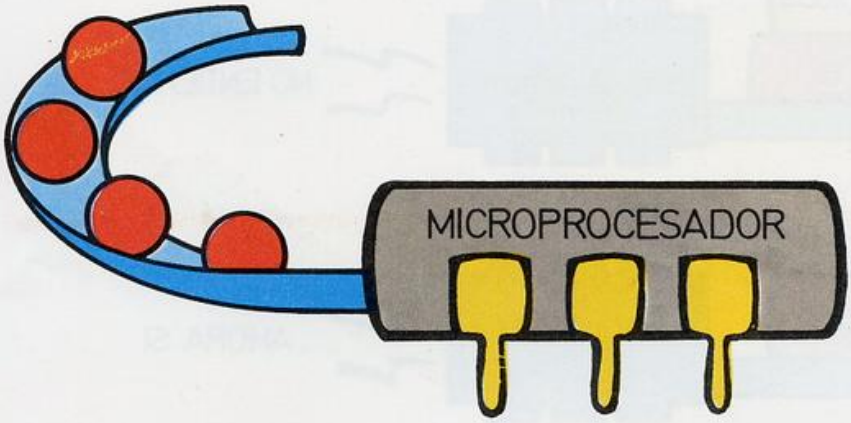
El empleo del código máquina conlleva dos importantes problemas: la dificultad de programación, y la imposibilidad de ejecución en otro microprocesador distinto de aquel para el cual fue concebido.

En cualquier caso, debemos considerar también que los 21 gráficos definidos por el usuario (UDG's) ocupan un total de 168 posiciones, por lo cual el límite superior efectivo de cada una de las memorias RAM será 32599 para el modelo de 16 K y 65367 para el de 48 K.
Si no vamos a necesitar los gráficos definidos por el usuario y deseamos almacenar una serie de bytes no superior a 168, cabe una interesante posibilidad, que es la de ocupar precisamente este bloque de bytes situado al final de la memoria RAM.
En este caso particular, no es necesario ejecutar ninguna sentencia **CLEAR** para protección del código almacenado, puesto que la zona de UDG's lo está permanentemente, incluso contra un **NEW**.
El siguiente programa almacena un código en la zona de gráficos definidos:

```
 10 REM - CARGADOR GRAFICOS (C) 1985 LO
PEZ MARTINEZ
 20 FOR I=0 TO 55
 30 READ J: POKE USR "A"+I,J
 40 NEXT I
 50 DATA 0,0,0,0,0,3,191,251
 60 DATA 0,0,24,60,60,24,0,0
 70 DATA 24,126,24,60,90,189,36,102
 80 DATA 153,90,126,60,24,24,60,102
 90 DATA 2,2,7,79,58,26,120,206
100 DATA 2,2,39,175,122,58,152,248
110 DATA 24,24,60,90,153,189,36,102
```

La línea 20 establece un bucle del número de bytes a leer (56) de las líneas **DATA** del final del programa. La línea 30 efectúa la lectura de las **DATA** y el **POKE** correspondiente en la posición siguiente, contando a partir del comienzo del área de UDG's **(USR "A")**. La línea 30 cierra el bucle. Por último, las líneas 50 a 110 contienen las series de bytes.
Hablando de un modo más general, podemos almacenar nuestro código en posiciones anteriores de memoria, protegidas esta vez con **CLEAR**, con sólo incluir algunas modificaciones en el programa anterior.
La forma de actuar es muy simple: primero debemos saber la longitud del código máquina a instalar, que restaremos del valor más alto de memoria posible descrito anteriormente, para efectuar un **CLEAR** especificando como parámetro la dirección decimal de memoria inmediatamente anterior. Hecho esto, efectuaremos los **POKE** correspondientes a partir de la primera dirección calculada.
El siguiente programa coloca al final de la memoria RAM, justo antes de los gráficos definidos

Un programa en código máquina es una sentencia de bytes que sólo adquieren sentido al ser interpretados por el microprocesador.



LET, INPUT, TO, IF,

En el lenguaje Ensamblador el programador no ha de recordar las instrucciones por su código numérico.

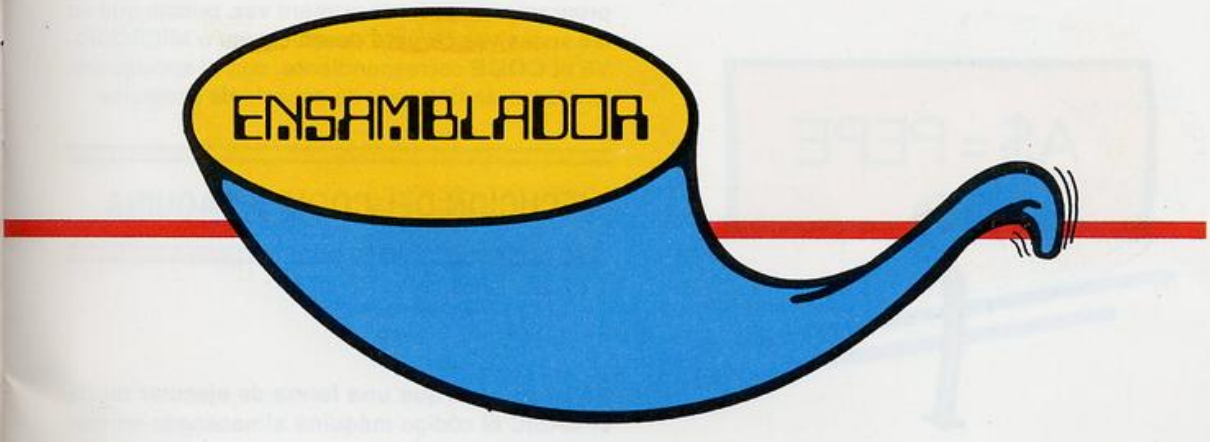
por el usuario, un programa en código máqina compuesto por una sucesión de 56 bytes:

```
 10 REM - CARGADOR  DATA (C) 1985 LOPEZ
MARTINEZ
 20 CLEAR 32543
 30 FOR I=0 TO 55
 40 READ J: POKE 32544+I,J
 50 NEXT I
 60 DATA 0,0,0,0,0,3,191,251
 70 DATA 0,0,24,60,60,24,0,0
 80 DATA 24,126,24,60,90,189,36,102
 90 DATA 153,90,126,60,24,24,60,102
100 DATA 2,2,7,79,58,26,120,206
110 DATA 2,2,39,175,122,58,152,248
120 DATA 24,24,60,90,153,189,36,102
```

En un intento de aproximar el lenguaje máquina a uno de alto nivel, se diseño el lenguaje Ensamblador.

La línea 20 efectúa un **CLEAR** a la dirección 32543 calculada como la más alta alcanzable en el modelo de 16 K (32767), restándole los 168 bytes ocupados por los gráficos definidos y los propios 56 bytes de la subrutina en cuestión (32767-168-56=32543).
Las líneas 30 a 50 se ocupan de situar, a partir de la primera dirección a continuación del **CLEAR**, los bytes que componen la subrutina, con ayuda de una estructura **FOR NEXT**. Por último, las líneas 60 a 120 contienen las **DATA** con los bytes de código máquina.
El sistema de almacenar bloques de bytes en sentencias **DATA** es sin duda eficaz, pero enormemente pesado cuando nos planteamos la introducción de un bloque más largo que desarrolle alguna subrutina compleja. En estos caso, es preferible recurrir a un programa cargador de código máquina el cual permita introducir los datos por medio de **INPUT**, así como corregir lo tecleado. El siguiente program es sólo una muestra de lo que puede ser un cargador de código máquina:

```
 10 REM - CARGADOR INPUT (C) 1985 LOPEZ
MARTINEZ
 20 LET B$=CHR$ 8+CHR$ 8+CHR$ 8+CHR$ 8+
"   "+CHR$ 8+CHR$ 8+CHR$ 8: DEF FN P$()=
("   "+A$)(LEN A$ TO )
 30 POKE 23658,8: BORDER 1: PAPER 1: IN
K 7: CLS
 40 INPUT "DIRECCION DE INICIO ?";A: LE
T P=A
 50 PRINT A;TAB 7;
 60 FOR I=1 TO 6
 70 INPUT "BYTE ?"; LINE A$
 80 IF A$="F" THEN  GO TO 60
 90 IF A$="E" THEN  GO TO 230
100 IF VAL A$<0 OR VAL A$>255 THEN  GO
TO 40
110 POKE A,VAL A$
120 LET A=A+1
130 PRINT FN P$();" ";
140 NEXT I: PRINT
150 GO TO 50
160 INPUT "NOMBRE DEL FICHERO ?";A$
170 IF A$="" THEN  GO TO 30
180 IF LEN A$>10 THEN  GO TO 160
190 SAVE A$CODE P,A-P+1
```

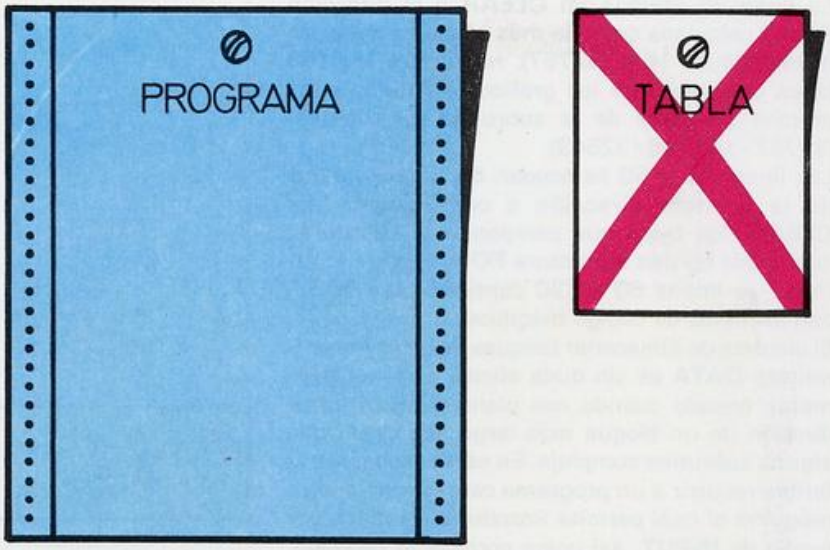


¡!

El almacenar bytes en sentencias **DATA** es enormemente pesado, por lo cual es preferible recurrir a un programa cargador, que permita introducir los datos mediante **INPUT**, así como posibilite la corrección de lo tecleado.

*

A través de la función **USR** podemos ejecutar desde el BASIC el código máquina almacenado en la memoria.



El lenguaje Ensamblador permite descifrar un listado sin ayuda de la tabla de códigos de operación.

```
 200 PRINT #0;"     PREPARE LA VERIFICAC
ION"
 210 VERIFY A$CODE P,A-P+1
 220 RUN
 230 IF I=1 THEN  PRINT AT 23-PEEK 23689
,31;: LET I=7
 240 PRINT B$;
 250 LET A=A-1: LET I=I-1
 260 GO TO 70
```

¡!

Para proteger un programa en código máquina, hemos de rebajar el RAMTOP (Variable del Sistema que indica la máxima dirección de memoria utilizable por el BASIC), situándolo por encima del texto BASIC y su área de variables.

La línea 20 define la cadena de caracteres **B$** formada por cursores hacia atrás **(CHR$ 8)** y tres espacios en blanco en su centro, al objeto de borrar la impresión del último byte tecleado y producir el correcto posicionamiento del cursor en la columna anterior (donde se detectó el error). Además, se define una función de usuario **P$( )**, cuyo objeto es colocar los datos numéricos en columnas y justificados por la derecha.

La línea 30 fija el teclado en mayúsculas y ajusta el color del **BORDER, PAPER** e **INK,** borrando la pantalla. La línea 40 solicita la dirección de inicio a partir de la cual ha de comenzarse la carga de bytes.

La línea 50 imprime, como cabecera de cada línea de pantalla, la primera de las direcciones sobre las cuales se está ejecutando la carga de la serie de seis bytes que vendrán a continuación.

Las líneas 60 a 140 gestionan la entrada de datos propiamente dicha. La primera línea del bloque se ocupa de aceptar los datos por medio de un **INPUT.** Las dos siguientes, del control de los caracteres significativos: **F** y **E.** El primero implica la conclusión de la carga y, el segundo, que se ha cometido un error en la introducción del byte de columna anterior, por lo cual se hace necesario eliminarlo y posicionar el cursor correctamente para permitir la nueva entrada del dato corregido.

El bloque incluye también la comprobación de que el dato tecleado se encuentra comprendido entre los valores admisibles (0 a 255), y ejecuta la sentencia **POKE** correspondiente para instalar el byte en el lugar que le corresponde, produciendo la salida a pantalla a seis columnas por línea.

Las líneas 160 a 220 gestionan la admisión y depuración del nombre que contendrá el bloque de bytes en cinta y efectúan el **SAVE** y **VERIFY** de eśtos.

Las líneas 230 a 260 corresponden a la subrutina de subsanación de errores, la cual permite no sólo eliminar y reemplazar físicamente el último dato tecleado, sino hacer lo propio con la impresión en pantalla, posicionando el cursor al comienzo de la columna anterior.

Debemos tener en cuenta que, a diferencia de lo que sucede con el código máquina almacenado en **DATAs** dentro de programa, es necesario efectuar la grabación del bloque de memoria en cinta una vez concluída la carga inicial.

De esta forma, sólo es necesario el empleo del programa cargador la primera vez, puesto que en las sucesivas cargará desde casete o MICRODRIVE el **CODE** correspondiente, con el consiguiente ahorro de tiempo y de espacio de programa.

Con el lenguaje Ensamblador se tiene la posibilidad de emplear nombres simbólicos («etiquetas») y «variables».

EJECUCION DEL CODIGO MAQUINA

No existe más que una forma de ejecutar desde el BASIC el código máquina almacenado en me-

moria y es a través de la función USR. USR tiene como argumento la dirección decimal de comienzo de la subrutina de la forma:

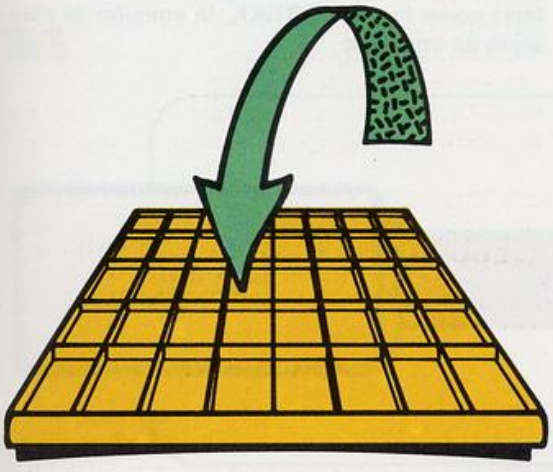
PRINT USR d

Donde **d** es la dirección decimal de comienzo de la subrutina y **PRINT** uno de los muchos métodos de emplear la función. Además de **PRINT**, puede utilizarse **RUN, RANDOMIZE, LET** y una variable numérica, etc...
A partir de la dirección especificada, la subrutina se ejecuta hasta encontrar un código de retorno (final de la subrutina), que se corresponde con el código 201 decimal, generalmente, aunque existen algunos otros códigos de retorno condicionales.
En este momento, se produce la vuelta al BASIC de forma similar a la de una subrutina BASIC al encontrar un **RETURN. USR** suministra a su retorno el valor del par registros **bc** del microprocesador, valor que podemos almacenar en una variable numérica para su posterior utilización.
De no ser necesario este valor, podemos recurrir a **RANDOMIZE,** por ejemplo, que no imprime nada como lo hace **PRINT** ni borra variables como **RUN.** Este sistema que es el más ampliamente difundido, tiene el inconveniente de afectar a los programas que se basan en la obtención de números aleatorios. Así pues, el sistema sustitutivo más inocuo es el de la asignación a una variable numérica; por ejemplo: **LET A=USR...**

Para proteger un programa en código máquina, debemos situarlo por encima del texto BASIC y su área de variables, haciendo descender el RAMTOP.

PRECAUCIONES EN EL USO DEL CODIGO MAQUINA

El Ensamblador se encarga de efectuar la traducción de lo codificado a código máquina y ubicarlo en el lugar de la memoria que le indiquemos.



En primer lugar hemos de ser conscientes de que el código máquina es un lenguaje como lo pueda ser el BASIC, aún más, mucho más natural, desde el punto de vista del Spectrum, que este último. Así pues, su manejo, aun indiscriminado, no puede producir ningún tipo de deterioro permanente en nuestro ordenador. Cualquier problema, por grave que parezca, se puede solucionar apagando y volviendo a encender el aparato.
Si no lo creemos, pongamos en funcionamiento el siguiente mini-programa, que se dedica a hacer **POKE** de valores aleatorios, en cualquier lugar de la memoria.

¡!

El lenguaje Ensamblador se diseñó para aproximar el lenguaje máquina a otro de mayor nivel.

*

El código máquina es el único lenguaje directamente comprensible por un microprocesador.

*

Debemos tener en cuenta que al apagar y encender el ordenador se borra por completo la memoria RAM.

DATA

El sistema de almacenar bloques de bytes en sentencias **DATA** *es eficaz, pero muy pesado.*

```
10 REM POKEADOR LOCO
20 POKE RND*65535, RND*255
30 GO TO 20
```

Ahora bien, el apagado y encendido, con la consoladora aparición del mensaje *(C) 1982 Sinclair Research Ltd.,* lleva como consecuencia el borrado completo de la memoria RAM. Así pues, supongamos que tras cinco horas delante del ordenador, tecleando un arduo programa, lo ponemos en marcha, y por un error en la introducción del código máquina, se produce un problema del tipo antes mencionado, conocido normalmente como «caída del Sistema».

Apagaremos y encenderemos, y el ordenador se encontrará en perfecto estado, pero los que no se encontrarán en un estado tan envidiable serán nuestros nervios, conscientes de haber arrojado a la basura cinco horas de pesado trabajo.

Así pues, nuestras precauciones deben ir encaminadas a no toparnos con desagradables sorpresas. La primera de ellas, debe ser no ejecutar nunca un programa que contenga una llamada al código máquina, sin haber grabado previamente una copia de seguridad de todo aquello que se pueda ver afectado por una posible «caída».

La segunda de las precauciones, es cerciorarse que antes de la ejecución del programa el código máquina necesario se encuentra presente en la memoria. Tan absurdo es intentar ejecutar un programa BASIC sin cargarlo con anterioridad, como uno en código máquina; con la sustanciosa diferencia de que en el primer caso sólo recibiremos un mensaje del Sistema, mientras que en el segundo, al no estar bajo el control de la ROM, la catástrofe software está prácticamente asegurada.

También es importante realizar la llamada al código máquina en el lugar preciso, del mismo modo que un programa BASIC no lo comenzamos a ejecutar desde cualquier punto, sino desde aquel que sabemos es su comienzo.

Por último, sólo muy contadas rutinas en código máquina, generalmente de poca longitud, son reubicables, es decir, funcionan cualquiera que sea el lugar de la memoria en el cual se carguen (respetando, lógicamente, su punto relativo de inicio, tal como se hace notar en el párrafo anterior). Por tanto, es muy importante situar el código máquina en el preciso lugar destinado a tal fin. Disponer, por ejemplo, de una rutina preparada para el modelo de 48 K e intentar utilizarla en uno de 16 K, no se limita a algo tan sencillo como cargarla en una posición válida en este último modelo, sino que precisa de modificaciones considerablemente más profundos.

Estas son las precauciones fundamentales a tener en cuenta, aunque nunca se nos debe olvidar que cuando entramos en una subrutina en código máquina, el Sistema pierde cualquier control, y éste pasa a establecerlo nuestro programa, por lo cual no nos podremos beneficiar de ventajas como la tecla **BREAK**, la emisión de mensajes de error, etc...

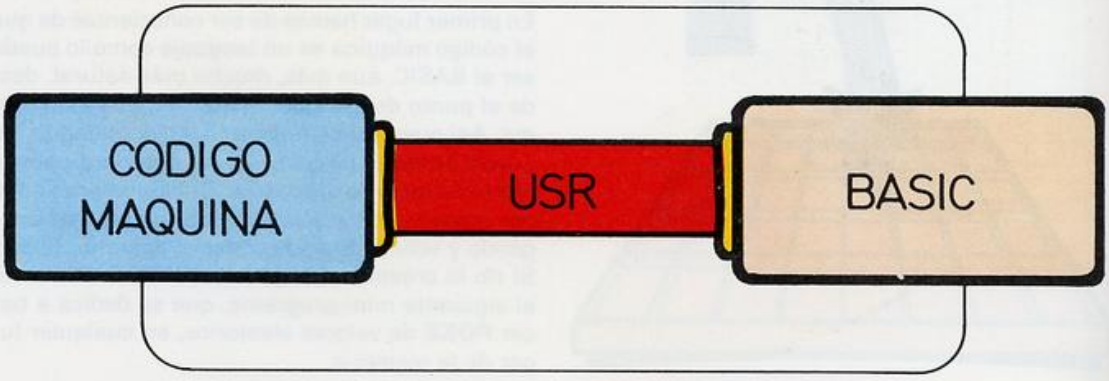
No existe más que una forma para ejecutar desde el BASIC el código máquina almacenado en memoria: **USR.**

¡!

El código máquina es el lenguaje comúnmente utilizado en los programas comerciales.

*

El código máquina es el lenguaje que menor cantidad de memoria necesita, y es incomparablemente más rápido que cualquier otro.



TU MICRO CONVERSA

ES una mañana cualquiera. Un molesto zumbido me saca súbitamente del quinto sueño, mientras alargo la mano hacia el teclado de mi Spectrum con la sana intención de detenerlo. Inmediatamente después, una fuerte voz comienza a sonar acusadora:

Ca-da dí-a tar-das más en des-per-tar-te. Lle-vo so-nan-do e-xac-ta-men-te cin-co mi-nu-tos y vein-ti-tres se-gun-dos. Son las sie-te ho-ras y do-ce mi-nu-tos. Vas con re-tra-so.

Por supuesto, los improperios con que le obsequio no pueden transcribirse, pero parecen hacerse inútiles cuando tercamente el ordenador me vuelve a colocar cada medio minuto la misma canción, recordándome imperturbable la hora que es. Malhumorado e irritado le grito: ¡CALLATE!

Aparentemente, levantarle la voz hizo su efecto. Esta vez, en tono más amable me pregunta: ¿Quie-res que en-ci-en-da las lu-ces?

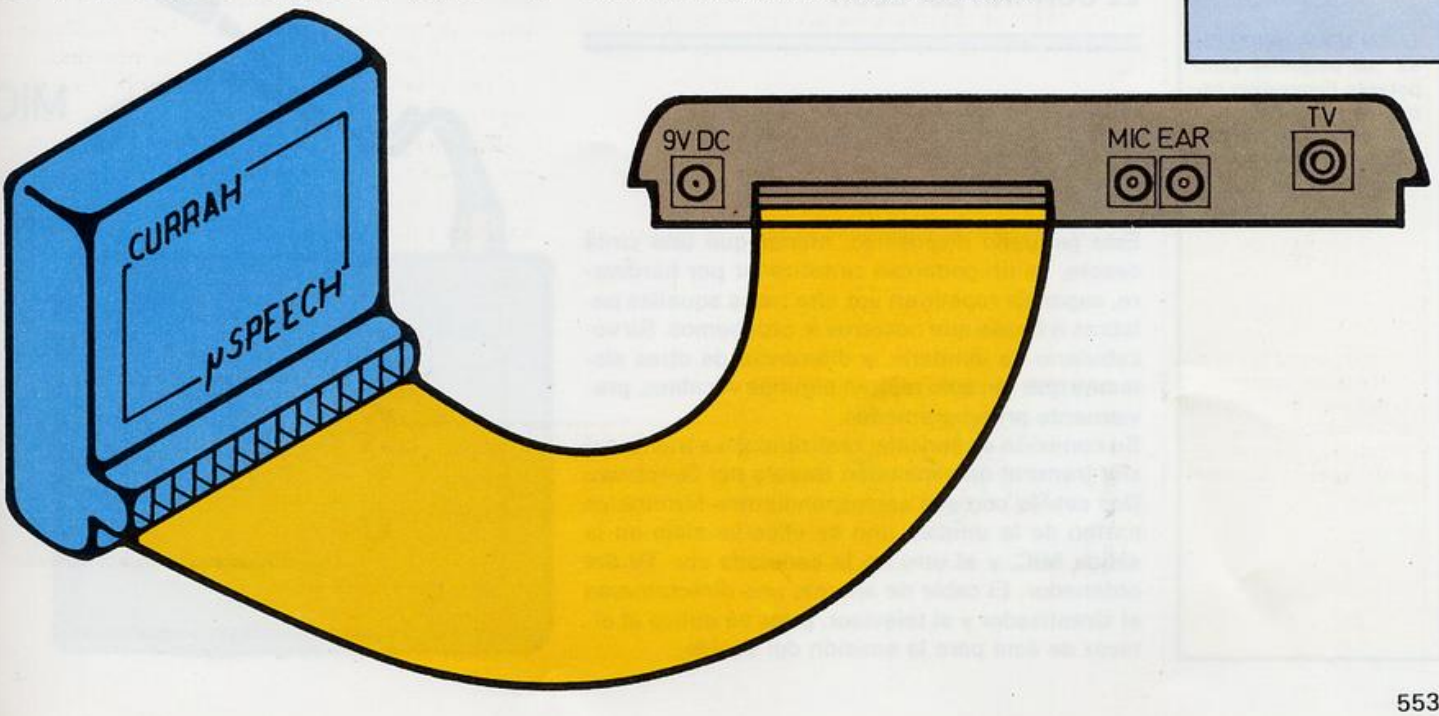
Su actitud resulta ahora mucho más amigable, así que para evitar males mayores contesto: sí, por favor. Tras ello, la lámpara de mi escritorio se ilumina y la radio comienza a sonar a tal volumen que ya no queda otro remedio que levantarse.

Antes de salir, intento comportarme con educación y me acerco a despedirme: ¡Adiós! —le digo—, pero sus ganas de hablar son inagotables y un tanto jocosamente me aconseja por último: Que no se te cru-cen los ca-bles. No dis-cu-tas con los pro-fe-so-res. Que ten-gas un bu-en dí-a. Te es-ta-ré es-pe-ran-do.

Finalmente, la pantalla del televisor se torna totalmente negra, y de esta manera, permanece en completo silencio.

Quizá pensemos que mantener un diálogo como el anterior con nuestro Spectrum, es cosa de ciencia ficción, pero la realidad es muy distinta, pues con ayuda de algunos dispositivos especiales, éste es capaz tanto de reconocer nuestra voz como de convertirse en un micro «parlanchín». Como ya adelantábamos en nuestro capítulo anterior, son varias las técnicas y periféricos encargados de potenciar las cualidades de este singular ordenador. Analicemos cuales son sus posibilidades.

La conexión del CURRAH µSPEECH al Spectrum se concreta a través del slot *de expansión trasero.*

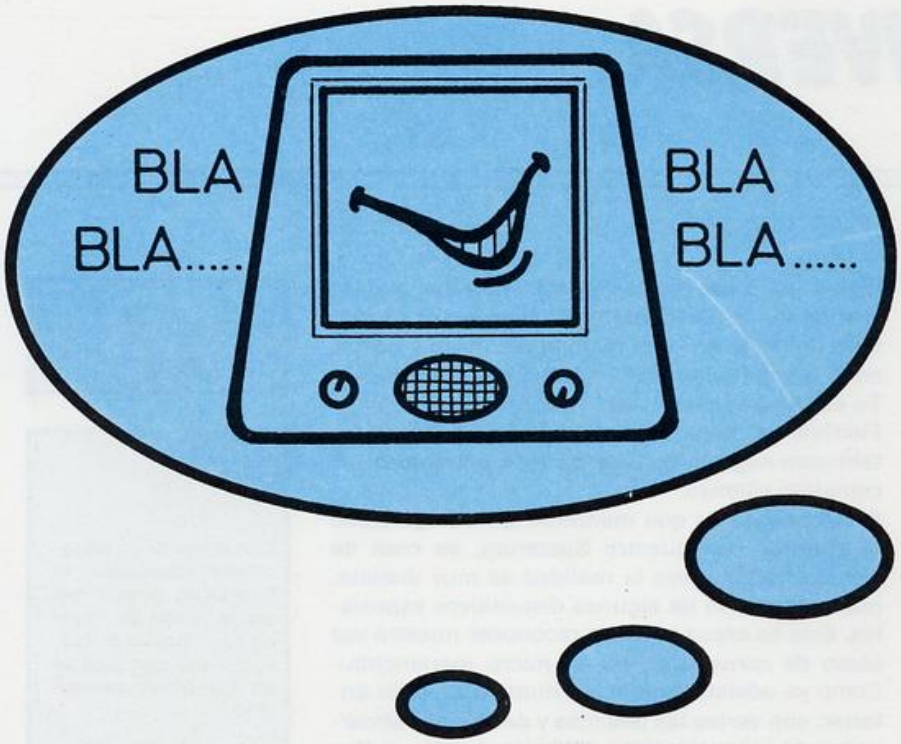


¡!

Con la ayuda de dispositivos especiales, el Spectrum puede ser capaz, tanto de reconocer nuestra voz, como de convertirse en un micro parlanchín.

*

El MICROCOMMAND es capaz de reconocer nuestra voz a través de un micrófono.



Mantener un diálogo con nuestro Spectrum no es cosa de ciencia ficción.

¡!

El CURRAH μSPEECH es un pequeño pero potente dispositivo capaz de repetir en alta voz todas aquellas palabras o frases que le ordenemos.

EL CURRAH μSPEECH

Este pequeño dispositivo, menor que una cinta casete, es un poderoso sintetizador por hardware, capaz de repetir en voz alta todas aquellas palabras o frases que nosotros le ordenemos. Su vocabulario es ilimitado, a diferencia de otros sistemas que tan solo repiten algunos vocablos, previamente preprogramados.

Su conexión es sencilla, realizándose a través del *slot* (ranura) de expansión trasera del Spectrum. Dos cables con sus correspondientes terminales parten de la unidad: uno de ellos se aloja en la salida MIC, y el otro en la señalada con TV del ordenador. El cable de antena, une directamente el sintetizador y el televisor, pues se utiliza el altavoz de éste para la emisión del sonido.

Como siempre, todas las operaciones de conexionado en el *slot* deben realizarse con la alimentación de corriente cortada, pues como sabemos, podemos ocasionar una grave avería. Una vez tomadas las precauciones oportunas estamos en condiciones de suministrar corriente al Sistema, con lo cual observaremos en la pantalla del televisor, además del mensaje habitual de Sinclair, el siguiente:

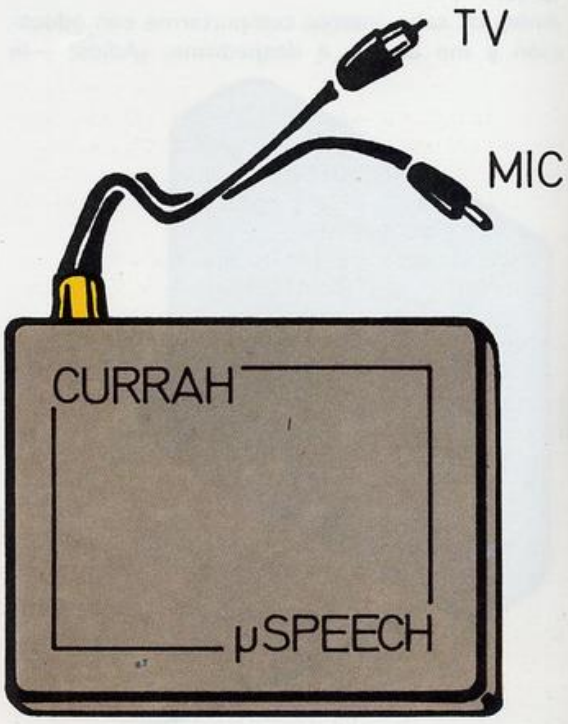
Speech System (c) CURRAH 1983

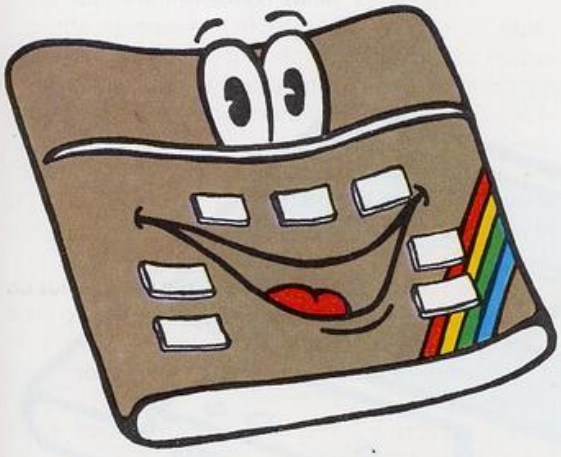
Este se encuentra ubicado en la parte superior de la pantalla, e indica que todo está preparado para comenzar a trabajar con el sintetizador, para lo cual, ajustaremos el volumen del televisor hacia la mitad de su recorrido.

Ahora pulsamos ENTER y el CURRAH lo pronunciará. Es posible que escuchemos el sonido distorsionado, y para solucionarlo actuaremos sobre un pequeño tornillo colocado en la parte superior de la unidad, hasta conseguir limpiarlo de interferencias.

Tras ello, pulsando cualquier tecla, el sintetizador ira repitiéndola, aunque si no le damos tiempo entre una y otra pulsación, o la mantenemos presionada, el Sistema de autorrepetición del teclado ocasionará una especie de indecisión en la

Dos cables parten de la unidad CURRAH μSPEECH: uno de ellos se aloja en la salida MIC y el otro en la señalada T.V.





El Spectrum, con ayuda de ciertos dispositivos, es capaz de convertirse en un micro parlanchín.

forma de hablar del CURRAH. Por ejemplo, si mantenemos pulsada la tecla DELETE se escuchará algo así como DE DE DE DELETE, hasta que finalmente liberemos la tecla.

Todo ello resulta de gran utilidad a la hora de introducir un programa, pues podemos escuchar todo cuanto vamos tecleando, sin necesidad de comprobar constantemente lo escrito en la pantalla del monitor.

No obstante, si deseamos que la unidad calle, bastará asignar a la variable reservada por el Sistema, KEYS, el valor 0. Para restablecer el sonido basta con introducir **LET keys=1**. Por otra parte, cuando ejecutemos un comando de inicialización **NEW**o **RANDOMIZE USR 0**, debemos pulsar **ENTER** para dejar preparado el sintetizador para su funcionamiento.

Cuando conectamos a la luz inicialmente la unidad, ésta mueve el RAMTOP del BASIC 256 bytes más abajo de su lugar habitual, es decir, crea un *buffer* de este tamaño a partir de la dirección 65111 en un Spectrum de 48 k (para el de 16 k habrá que restar 32767), donde almacenar los datos sobre las voces a pronunciar.

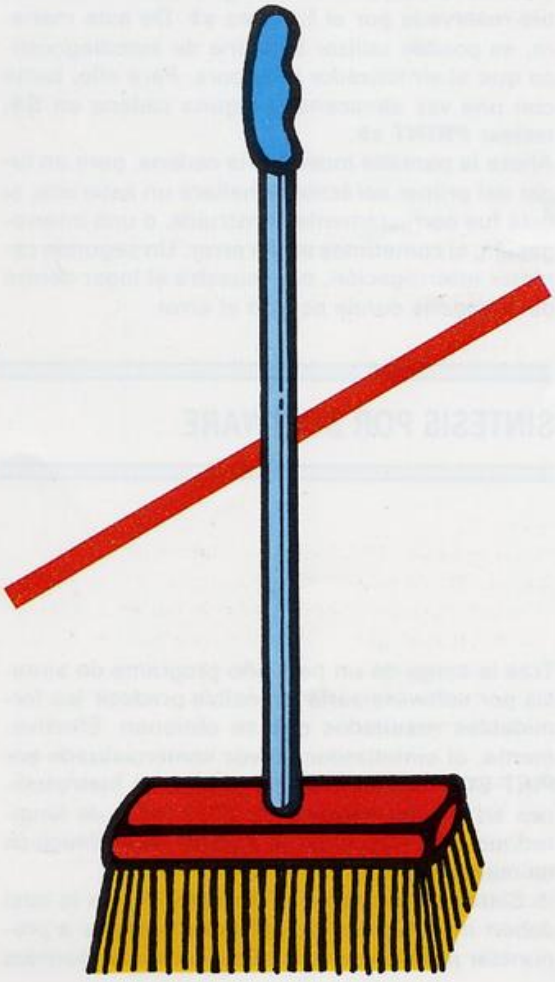
Este puede ser modificado mediante el uso de **CLEAR**, pero si lo intentamos con un valor por encima de dicha dirección, el Sistema comienza una secuencia de inicialización. Igualmente, podemos hacerlo más pequeño, pero cualquier frase que ocupara mayor longitud de la reservada, sería completamente ignorada.

El CURRAH maneja alófonos, es decir, representaciones de las unidades fonéticas mínimas, procedentes de la lengua inglesa. Por ello, sólo la práctica o el conocimiento de este idioma, facilitan la correcta construcción de las distintas palabras o frases en Castellano.

Los símbolos alófonos están divididos en dos grupos: los que constan de un solo carácter y los compuestos por varios de ellos encerrados entre paréntesis. Por ejemplo, la sintaxis correcta para que el Sistema pronuncie el comando **NEW** es *ny(ouu)*.

Una particularidad es la posibilidad de introducción de las letras tanto minúsculas como mayúsculas, con lo cual se consigue modificar la entonación con la que el Sistema pronuncia la palabra. En un alófono entre paréntesis, es el último carácter de éste el que marca la entonación, es decir (aAa) no tiene entonación, mientras que (aaA), sí la tiene.

Para limpiar las interferencias de sonido en el CURRAH, actuaremos sobre un pequeño tornillo situado en la parte superior de la unidad.



¡!

El vocabulario del CURRAH μSPEECH es ilimitado.

*

La conexión del CURRAH a nuestro Spectrum ha de realizarse con la alimentación de corriente cortada; de otra forma, podríamos ocasionar una grave avería.

*

La síntesis de voz por software a diferencia de la hardware, no tiene la posibilidad de modificar la entonación manejando las mayúsculas o minúsculas.

¡OLÉ!

EL CURRAH µSPEECH maneja alófonos (representaciones de las unidades fonéticas mínimas) procedentes de la lengua inglesa.

¡!

El CURRAH resulta de gran utilidad a la hora de introducir programas, pues podemos escuchar lo que tecleamos sin necesidad de mirar a la pantalla.

Para que el CURRAH repita las frases que nosotros deseamos, debemos asignar éstas a la variable reservada por el Sistema **s$**. De esta manera, es posible utilizar la rutina de autodiagnóstico que el sintetizador incorpora. Para ello, basta con una vez almacenada alguna cadena en **S$**, teclear **PRINT s$**.
Ahora la pantalla mostrará la cadena, pero en lugar del primer carácter se hallará un asterisco, si ésta fue correctamente construida, o una interrogación, si cometimos algún error. Un segundo carácter interrogación, nos muestra el lugar dentro de la cadena donde ocurrió el error.

SINTESIS POR SOFTWARE

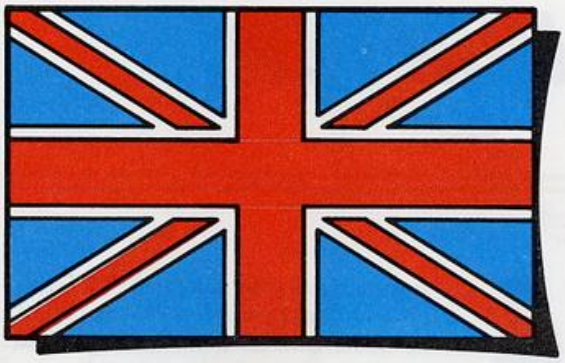
Tras la carga de un pequeño programa de síntesis por software sería imposible predecir los formidables resultados que se obtienen. Efectivamente, el sintetizador de voz comercializado por PINT SOFT consiste en un bloque de instrucciones en código máquina de 2222 bytes de longitud, ubicada inicialmente a partir de la dirección de memoria 63000.
El Sistema se reserva la variable **V$**, en la cual deben introducirse las palabras o frases a pronunciar por el ordenador. Este utiliza los sonidos castellanos, es decir, cada palabra se pronuncia de la misma forma en la cual se escribe, salvo unas pocas excepciones como pueden ser la Ñ (se emplea GN para imitarla), o la LL (escribiremos LY).
La salida de la voz se efectúa a través del pequeño minialtavoz del Spectrum, por ello, resulta útil emplear alguno de los sistemas comentados en el capítulo anterior, con objeto de amplificar el sonido.
Para utilizar este digitalizador de voz en nuestros programas, bastará con cargar la rutina en código máquina en la memoria, y desarrollar unas sencillas instrucciones encaminadas a asignar a la variable **V$** las voces deseadas. Por ejemplo, un programa basado en este sistema podría tener el siguiente aspecto:

```
 5 CLEAR 62999:LOAD "" CODE
10 INPUT "Frase a pronunciar ";V$
20 RANDOMIZE USR 63000
30 GOTO 10
```

A diferencia con el CURRAH, no tenemos la posibilidad de modificar la entonación en las palabras según manejemos mayúsculas o minúsculas, aunque podemos utilizar la repetición de vocablos, con objeto de matizar su pronunciación.
La rutina sonora es completamente reubicable, por lo cual los usuarios del Spectrum de 16 K, tras unas sencillas modificaciones, pueden tener acceso a ella e incorporarla a sus propios programas.

EL MICRO COMMAND

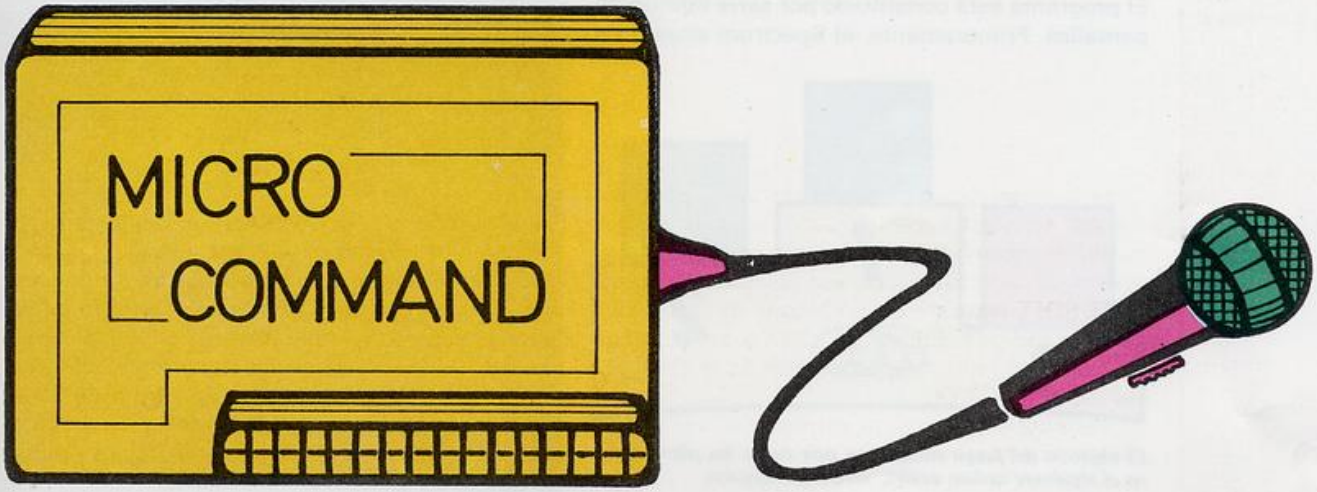
Si sorprendente puede resultar escuchar la voz de nuestro ordenador, más lo es comprobar como también está facultado para reconocer la nuestra. Las técnicas desarrolladas hasta el momento, no permiten disponer de un vocabulario ilimitado como en el caso de los sintetizadores de voz, pues la ocupación de memoria es mucho mayor.
El MICRO COMMAND es un interface reconocedor de voz que se conecta en el port de expansión del Spectrum. Un micrófono destinado a recoger nuestras órdenes, se acopla en la parte superior del dispositivo.
Una cinta que se suministra con el Sistema, además de ciertos programas de demostración, contiene las utilidades necesarias para hacer un uso correcto de éste. Primeramente, es imprescindible seguir un proceso de enseñanza donde repetimos las palabras que deseamos sean reconocidas (hasta un máximo de quince), ante el micrófono.
Seguidamente, conviene cerciorarse que todo fue bien y para ello, el programa va solicitándolas y debemos repetirlas hasta conseguir un alto porcentaje de fiabilidad. Tras ello, ya estamos en condiciones de incorporarlas a un programa propio.
Cada palabra se almacena en la memoria como un número, y es a éste al cual deberemos hacer referencia cada vez que solicitemos una entrada sonora. Es conveniente no emplear palabras largas, pues podría darse el caso que la unidad tuviera ciertos problemas para reconocerlas, si hubiera dos o más con los mismos sonidos al principio.



El sintetizador de voz por software comercializado por PIN SOFT utiliza sonidos castellanos.

El MICRO COMMAND sigue dos métodos intercambiables para escuchar las palabras: uno rápido y otro lento. Si utilizamos el primer método deberemos indicarle al Sistema el tiempo que debe esperar antes de prepararse para recibir una nueva palabra, pues de no hacerlo, considerará como una palabra completa la última mitad de la anterior.
En resumen, si todavía alguno de vosotros alberga dudas sobre la posibilidad de mantener un diálogo con nuestro micro, no tiene más que añadirle programas o periféricos como los descritos anteriormente, y casi con toda seguridad, no quedará defraudado.

El MICRO COMMAND es un interface reconocedor de voz que consta de un micrófono, y una cinta con el software necesario para su uso.



¡!

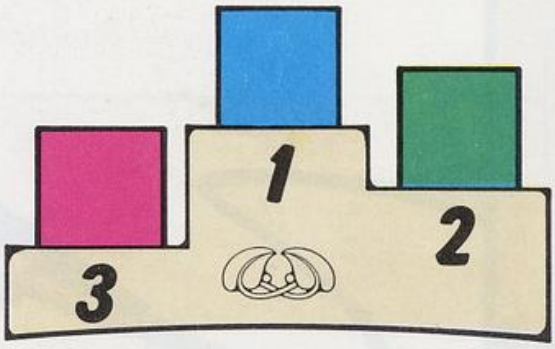
Para la óptima construcción de palabras castellanas en el CURRAH precisamos una cierta práctica o conocimiento de la lengua inglesa, pues son sus alófonos los empleados por este sintetizador.

HUNTER

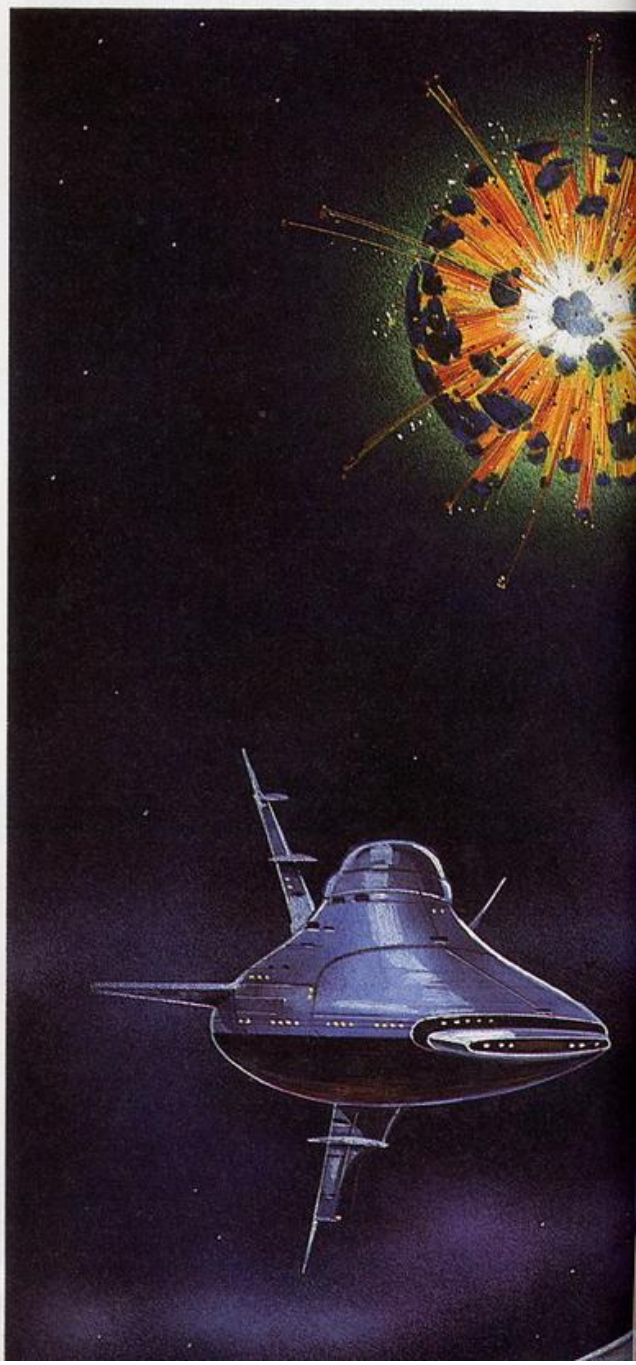
La gran nave interestelar S.H.1 cruza en estos momentos el sistema solar de SHADOM, perteneciente a la galaxia 37, con dirección a la Tierra.
Los tripulantes de la SPACE HUNTER 1 se encuentran en situación crítica debido a una rotura en el tanque principal de combustible. Los gases emanados por el combustible, al ponerse en contacto con el aire de la nave, han provocado la oxidación del sistema de turbopropulsión y como consecuencia el sobrecalentamiento de los bidones de cerveza, haciendo esta absolutamente imbebible.
A causa de estos problemas, el comandante de la nave se ve obligado a conectar la computadora de rastreo para la localización de planetas habitados, con el fin de reponer combustible y cerveza para poder continuar el viaje de regreso a la Tierra.
Como ya podemos suponer esta, aparentemente, sencilla misión no lo es tanto, y le va a dar más de un dolor de cabeza a nuestro capitán, al margen de los propios de la resaca de la cerveza...

EL PROGRAMA

El programa está constituido por serie infinita de pantallas. Primeramente, el Spectrum situará de forma aleatoria en la pantalla cinco planetas, simbolizados por cuadritos de tres colores diferentes.
La nave está representada por un pixel que irá dejando marcada en la pantalla su trayectoria, un inconfundible rastro de energía que deberemos evitar.



El objetivo del juego consiste en unir todos los planetas en el siguiente orden: azules, verdes y morados.



El objetivo del juego consiste en unir todos y cada uno de los planetas en el siguiente orden: azules, verdes y morados. En el caso en que este orden fuese roto, nuestro Spectrum respondería con un zumbido, i .dicando que hemos deshecho la secuencia, y por tanto nos vemos en la obligación de volver a empezar.

Por cada pantalla superada, el ordenador añadirá tres planetas más para la siguiente, aumentando así la dificultad del juego.

Es muy importante recordar que si en nuestro trayecto se nos ocurriese cruzar la estela de energía que vamos trazando, pereceríamos instantaneamente.

Por cada pantalla superada, el ordenador añadirá tres planetas más, aumentando así el nivel de dificultad.

Para el control de los mandos del SPACE HUNTER 1, utilizaremos las siguientes teclas:

Q - Arriba
A - Abajo
O - Izquierda
P - Derecha

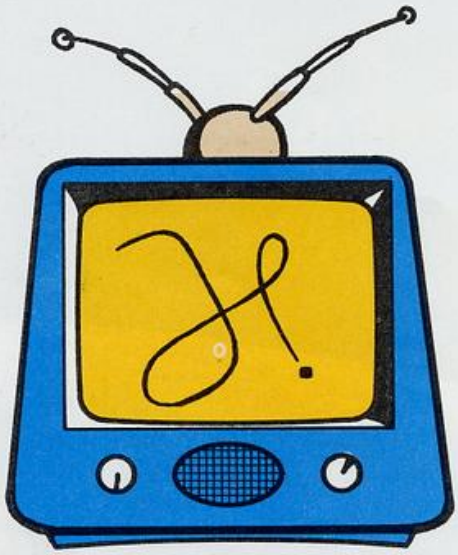
El programa únicamente incluye un gráfico definido (de la tecla A), que aparece subrayado según la notación habitual en la línea 260. Así pues, una vez introducido el programa, sólo hemos de ejecutar **SAVE "S.HUNTER" o SAVE "S.HUNTER" LINE 10** para la grabación del programa en modo normal o autoejecución, respectivamente.

¡!

Para conseguir pasar a otra pantalla es necesario unir todos los planetas que aparezcan en el siguiente orden:

1 - Planetas azules
2 - Planetas verdes
3 - Planetas morados

La SPACE HUNTER 1 está representada por un pixel que irá dejando marcada su trayectoria en la pantalla.



```
 10 REM ************************
 20 REM * J.M.MAYORAL SERRANO *
 30 REM ************************
 40 REM * SPACE HUNTER © 1985 *
 50 REM ************************
 60 LET PANT=5: POKE 23658,8
 70 LET PA=1: LET P=0
 80 GO SUB 800
 90 PAPER 0: BORDER 0: CLS
100 INK 9
110 RANDOMIZE
120 DEF FN W(X,Y)=10*(6-ATTR (INT ((175-Y)/8),INT (X
/8)))
130 LET CONT=0
140 PLOT 0,167
150 DRAW 255,0
160 DRAW 0,-159
170 DRAW -255,0
180 DRAW 0,159
190 PRINT AT 21,0;"PANTALLA ";PA
200 FOR Q=1 TO PANT
210 LET C=INT (RND*3)+3
220 LET X1=INT (RND*17)+2
230 LET Y1=INT (RND*27)+2
240 IF X1>=10 AND X1<=11 AND Y1>=14 AND Y1<=20 THEN
BEEP .05,40: GO TO 220
250 IF ATTR (X1,Y1)>=3 AND ATTR (X1,Y1)<=5 THEN  GO
TO 220
260 PRINT AT X1,Y1; INK C; INVERSE 1;"A"
270 NEXT Q
280 LET X=112
290 LET Y=88
300 LET P$="P"
310 LET S2=0
320 LET S1=0
330 PRINT AT 21,15;"PUNTOS = "
340 PRINT  INK 5; BRIGHT 1;AT 21,24;P
350 PLOT X,Y
360 LET K$=INKEY$
370 IF K$="" THEN  LET K$=P$: GO TO 390
380 LET P$=K$
390 LET Y=Y+(K$="Q")-(K$="A")
400 LET X=X+(K$="P")-(K$="O")
410 IF NOT POINT (X,Y) THEN  GO TO 350
420 IF FN W(X,Y)=-10 THEN  GO TO 510
430 LET S1=FN W(X,Y)
440 IF S1>=S2 THEN  LET S2=S1: LET P=P+S1: PRINT AT
21,24; INK 5;P: LET CONT=CONT+1: BEEP .03,10: GO TO 4
60
450 GO TO 510
460 PLOT X+(4 AND K$="P")-(4 AND K$="O"),Y+(4 AND K$
="Q")-(4 AND K$="A")
470 LET X=X+(8 AND K$="P")-(8 AND K$="O")
480 LET Y=Y+(8 AND K$="Q")-(8 AND K$="A")
490 IF CONT=PANT THEN  GO TO 0570
500 GO TO 350
510 BEEP 2,-10
520 PRINT )0; INVERSE 1;"  PULSA UNA TECLA PARA EMPE
ZAR  "
530 LET K$=INKEY$
540 IF K$="" THEN  GO TO 530
550 BEEP .1,40
560 RUN
570 LET PANT=PANT+3
580 LET PA=PA+1
590 LET W$="  E S T U P E N D O  "
600 PAPER 2: INK 6
610 LET FL=1: LET COL=4
620 GO SUB 690
630 LET COL=1: LET FL=0
640 PAPER 5: INK 1
650 LET W$="PASO A LA SIGUIENTE PANTALLA"
660 PRINT  PAPER 0;AT 0,0;"
                           "
670 GO SUB 690
680 GO TO 740
690 FOR N=1 TO LEN W$
700 PRINT AT 0,COL+N; FLASH FL;W$(N)
710 BEEP .05,40
720 NEXT N
730 FOR W=1 TO 50: NEXT W: RETURN
740 FOR N=1 TO 20
750 BEEP .03,10: BEEP .03,30
760 NEXT N
770 PRINT  PAPER 0;AT 0,0;"
                           "
780 BEEP .2,40
790 GO TO 90
800 REM  GRAFICOS USUARIO
810 DATA 0,126,102,66,66,102,126,0
820 FOR F=0 TO 7
830 READ A
840 POKE USR "A"+F,A
850 NEXT F
860 RETURN
```