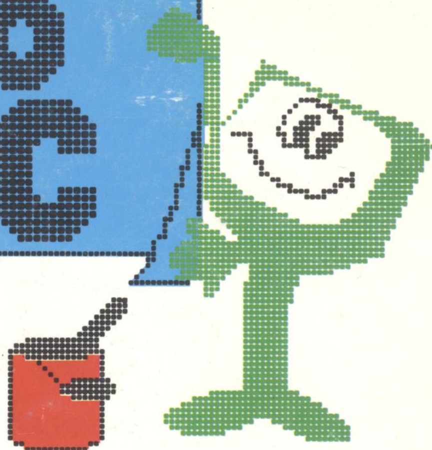
VIDEO BASIC



20 LECCIONES DE BASIC
PARA APRENDER CON EL C-64

INGELEK

JACKSON

- CPU: organización externa e interna
- Los BUSES
- El código máquina
- Ventajas y desventajas del Assembler
- SYS, USR
- Memorización de programas en código máquina
- Videoejercicios
- Videojuego N.º 18

18

COMMODORE
C-64

VIDEO BASIC
Una publicación de
INGELEK JACKSON
Director editor por INGELEK:
Antonio M. Ferrer
Director editor por JACKSON HISPANIA:
Lorenzo Bertagnolio
Director de producción:
Vicente Robles
Autor: Softidea
Redacción software italiano:
Francesço Franceschini,
Stefano Cremonesi
Redacción software castellano:
Fernando López, Antonio Carvajal,
Alberto Caffarato, Pilar Manzanera
Diseño gráfico:
Studio Nuovaidea
Ilustraciones:
Cinzia Ferrari, Silvano Scolari,
Equipo Galata
Ediciones INGELEK, S. A.
Dirección, redacción y administración,
números atrasados y suscripciones:
Avda. Alfonso XIII, 141
28016 Madrid. Tel. 2505820
Fotocomposición: Espacio y Punto, S. A.
Imprime: Gráficas Reunidas, S. A.

Reservados todos los derechos de reproducción y publicación de diseño, fotografía y textos.
©Grupo Editorial Jackson 1985.
©Ediciones Ingelek 1985.
ISBN del tomo 4: 84-85831-23-3
ISBN del fascículo: 84-85831-14-4
ISBN de la obra completa: 84-85831-13-6
Depósito Legal: M-15075-1985
Plan general de la obra:
20 fascículos y 20 casetes, de aparición quincenal, coleccionables en 5 estuches.
Distribución en España:
COEDIS, S. A.
Valencia, 245. 08007 Barcelona.
INGELEK JACKSON garantiza la publicación de todos los fascículos y casetes que componen esta obra y el suministro de cualquier número atrasado o estuche mientras dure la publicación y hasta un año después de terminada.
El editor se reserva el derecho de modificar el precio de venta del fascículo, en el transcurso de la obra, si las circunstancias del mercado así lo exigen.
Diciembre, 1985
Impreso en España.

INGELEK

JACKSON

SUMARIO

HARDWARE 2
La CPU. Organización externa de una CPU. Los buses.

EL LENGUAJE 10
El código máquina. Desventajas y ventajas del Assembler.
USR, SYS.

LA PROGRAMACION 22
Descubriendo el Assembler. La numeración hexadecimal. Cómo memorizar los programas en código máquina. Ejemplos Assembler.
Lenguaje máquina para imprimir donde quieras.

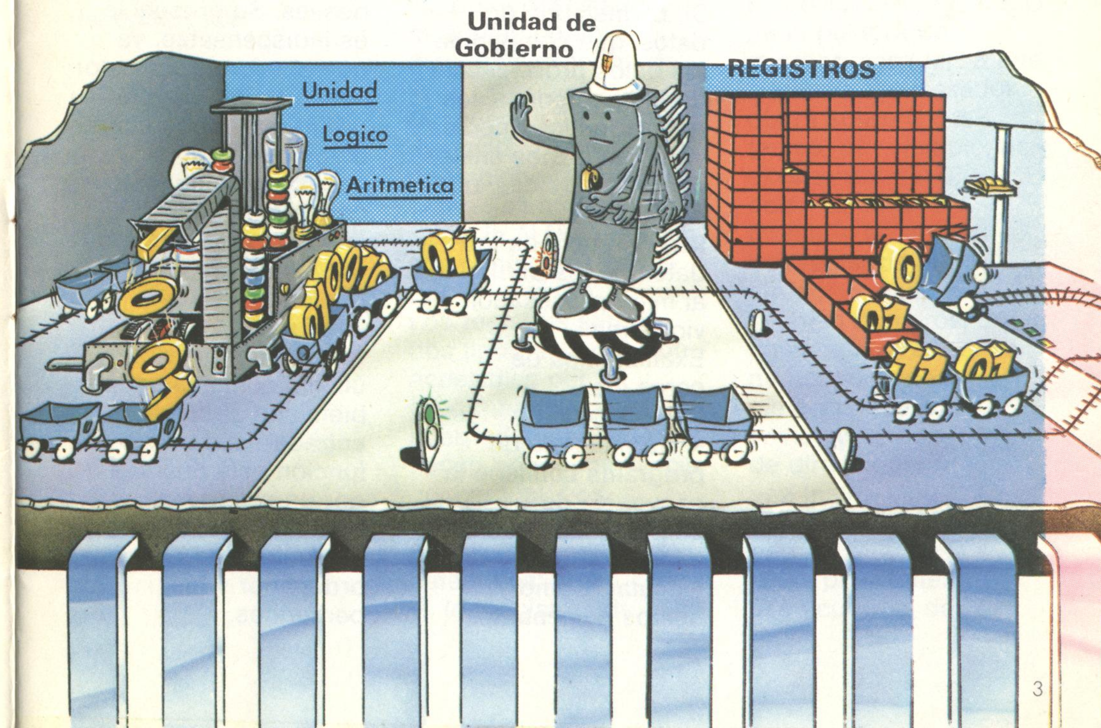
VIDEOEJERCICIOS 32

Introducción

Antes de continuar... demos algunos pasos hacia atrás. Para presentar el temido código máquina (C/M para quien quiera presumir) y el igualmente conocido assembler, es necesario refrescar los conceptos base de la CPU, su organización externa e interna, los buses.
No se trata de un inútil ejercicio académico; para obtener sorprendentes prestaciones del propio ordenador, es necesario —a veces imprescindible, cuando se quiere ahorrar memoria y tiempo de ejecución— programar directamente el micropresador que lo gobierna.

La CPU

El término CPU se usa muy a menudo con dos acepciones distintas. Considerando el ordenador dotado de periféricos, se designa a veces como CPU a la parte que contiene la unidad central propiamente dicha (aquélla que ejecuta las instrucciones en la secuencia correcta), la memoria central y los interfaces de conexión con el resto del ordenador (y, en muchos personales, también con el teclado). En este caso CPU o unidad central es sinónimo de: «el ordenador entero y verdadero, excluidos los periféricos físicamente separados de él (pantalla, grabadora, impresora, etc.)».
En cambio, desde el punto de vista lógico, la CPU es solamente la unidad central de cálculo y control, excluyendo, por tanto, memorias e interfaces de cualquier tipo. Dado que nosotros siempre habíamos entendido el término «CPU» (y también el término «unidad central») solamente en esta segunda acepción, lo seguiremos haciendo a lo largo de todos nuestros capítulos.
Hecha esta debida precisión, entremos rápidamente en el meollo del tema de esta lección, es decir, la descripción pormenorizada de la CPU.
Hasta ahora, siempre habíamos acudido a la unidad central como a una especie de «caja mágica», a la cual hacíamos preguntas en entrada (siguiendo una sintaxis y una gramática determinadas) y de la cual obteníamos en salida las respuestas.
Este es el momento preciso de levantar esta limitación, tratando de penetrar en ella con mayor profundidad. El conocimiento —más o menos profundo— de la unidad central no suele exigírsele generalmente al programador; aún más, en los límites de lo posible, todos los lenguajes de alto nivel (como el BASIC) tratan de evitar que existan ataduras de interdependencia entre el hardware y el software.
La fase de escritura de un programa (por lo menos en un lenguaje de alto nivel) debe, por tanto, ser siempre desarrollada con absoluta autonomía de las características hardware del ordenador utilizado, de tal forma que se resienta lo menos posible de las peculiaridades constructivas típicas de un sistema determinado.
El BASIC, como sabes bien —y sobre todo a causa de varios motivos históricos que han provocado una evolución anómala— no puede ciertamente ser considerado en absoluto como el lenguaje más «transportable» y «compatible» (basta echar una ojeada a un idéntico programa escrito para dos ordenadores diferentes para darse cuenta de esto inmediatamente).
Sin embargo, en los últimos años han aparecido muchos otros lenguajes de alto nivel, que permiten alcanzar el objetivo de la transportabilidad (es decir, de la casi perfecta compatibilidad entre máquinas distintas) casi completamente.
No obstante, en determinadas circunstancias, sobre todo en los casos de utilización especialmente «evolucionada» del ordenador o de programación en lenguajes de bajo nivel (es decir, mucho más próximos a la máquina que al hombre) puede ser, en cambio, de gran utilidad hacer referencia específica a un ordenador determinado y —en consecuencia— a un microprocesador específico. Además, a nivel de cultura general, el conocimiento de la constitución de la unidad central —por muy simplificada o limitada que sea— puede resultar un complemento necesario o incluso imprescindible.



Organización externa de una CPU

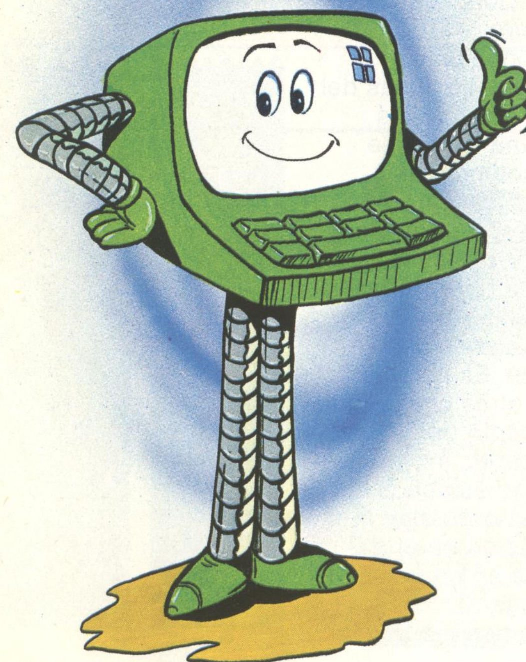
Un microprocesador no sabe hacer nada solo. Para que pueda trabajar es necesario conectarle las memorias y los interfaces. Una vez conectado a estos circuitos el microprocesador se convierte en un microordenador, generalmente divisible en cuatro partes diferentes, que comunican unas con otras a través de un bus de datos, un bus de direcciones y un bus de control (por el momento, puedes asociar a la palabra «bus» la correspondiente castellana «conexión»).
Estas cuatro partes son:
1) El microprocesador.
2) La memoria de programa, que indica a la CPU las funciones a cumplir. Puede ser —según los casos— memoria ROM o memoria RAM.
3) La memoria de datos, que compagina los datos provenientes de los periféricos, los resultados intermedios y los resultados finales.
4) Los interfaces, que permiten la transferencia de los datos de los periféricos al microprocesador, y viceversa.
Examinemos de cerca los tres últimos.
— La memoria de programa contiene la secuencia de instrucciones que el microprocesador debe ejecutar. Como ya hemos comentado, puede ser de tipo RAM o ROM. Cuando por ejemplo tú enciendes tu ordenador el rótulo que aparece en la pantalla está gestionado en la CPU por un programa que se encuentra en la ROM, mientras que cuando escribes una instrucción mediante el teclado ésta se memoriza en la RAM.
— La memoria de datos es, en cambio, un tipo de memoria notablemente distinta de la memoria de programa. Es a la fuerza una memoria RAM, dado que tiene que poder ser leída o escrita todas la veces que lo desees. Su presencia es indispensable, ya que el microprocesador —para poder ejecutar su programa— debe utilizar datos que, necesariamente, estén contenidos en la memoria de datos o que provengan de los periféricos a través de interfaces.
— Los interfaces, por último, como ya sabes bien, son circuitos de entrada-salida cuya función está definida por un programa. Permiten la comunicación entre el ordenador y los periféricos.

Organización interna de una CPU

Un microprocesador contiene bastantes miles de transistores y de otros componentes electrónicos: no es este el lugar de describirlo con grandes detalles (y por otra parte, esto tampoco sería de gran utilidad).
En cualquier caso, un microprocesador está constituido por tres partes principales:
— la unidad de control
— la unidad aritmético-lógica
— los registros.
La unidad de control decodifica las instrucciones que son tomadas de la memoria de programa y elabora las señales de comando necesarias para la ejecución de una instrucción.
La función de la unidad artimético-lógica (también llamada abreviadamente ALU) consiste, en cambio, en la ejecución de las operaciones lógicas y aritméticas sobre los datos que la alimentan a través de sus dos puertas de entrada, que son, respectivamente, «la entrada izquierda» y «la entrada derecha»: estas puertas se pueden imaginar como las dos extremidades más altas de un diagrama en forma de «V».
Después de la ejecución de una operación aritmética, como una suma o una resta, la ALU hace salir sus contenidos por la punta de la «V».
Los registros son de dos tipos: algunos son accesibles por programa, otros son internos de la CPU y no tienen gran importancia conceptual. Los registros accesibles se dividen en tres categorías:
— los registros de datos
— los registros de direcciones
— el contador de programa (Program Counter), el puntero del stack (pila) y el registro de estado.
Los registros de datos son en la práctica localizaciones de memoria que permiten la memorización temporal de las informaciones durante sus desplazamientos entre la unidad aritmético-lógica, las memorias y los interfaces.
Su longitud, es decir, el número de bits que componen cada registro es obviamente igual a la de la palabra del microprocesador: 8 bits, si el microprocesador opera sobre 8 bits (como, por ejemplo, es el caso del 6510, es decir, de la CPU montada en el C64).
Los registros de dirección, llamados también punteros, contienen direcciones de las posiciones de memoria que son enviadas al bus de dirección con una instrucción determinada que permita el acceso a estas posiciones.
¿Te acuerdas de

cuando –hablando de las memorias– comentabamos al respecto que cada localización disponía de una dirección bien precisa? Bien, esta dirección sirve a los registros de dirección para hacer que la CPU pueda referirse a cualquier posición de la memoria.
El puntero del stack es un registro de direccionamiento especial que señala a una determinada zona de la memoria, llamada «área del stack». Se decrementa automáticamente después de cada transferencia de una información a esta zona, y se incrementa después de cada remoción. El puntero del stack, como ya hemos podido comentar en una de las lecciones pasadas, tiene una función especialmente importante en la llamada y en el retorno de los subprogramas.
El Program Counter (o contador de instrucciones) vigila la ejecución de todo el programa.
Inicialmente se carga con la dirección de la primera instrucción del programa, y, a continuación, señala al microprocesador las direcciones de las instrucciones que deben ser ejecutadas sucesivamente.
Mientras el microprocesador lee en la memoria de programa una instrucción, y la ejecuta, el contador prosigue hasta la dirección de la siguiente instrucción y así sucesivamente.
Generalmente el microprocesador ejecuta las instrucciones en orden secuencial, es decir, una detrás de otra. Sin embargo, en el caso de algunas instrucciones, la ejecución secuencial del programa puede ser modificada: en el Program Counter se carga entonces la dirección de salto, y se conserva la eventual dirección de retorno al programa principal para ser utilizada a continuación.
El registro de estado contiene en cambio un cierto número de bits puestos a 0 o a 1, según hayan sido verificadas determinadas condiciones después de la ejecución de algunas instrucciones (por ejemplo, si el resultado es positivo o negativo, con resto o sin él, etc.).
El desarrollo del programa puede ser modificado en función de los valores tomados por uno o más bits del registro de estado. Es uno de los registros más importantes que el programador tiene a su disposición. Los registros pueden ser utilizados para diversas comprobaciones especiales o condiciones excepcionales, o también para verificar velozmente algunos resultados equivocados.
Existe finalmente otro

importantísimo registro que equipa a la ALU: el acumulador. Este es siempre una de las dos entradas de la ALU (poco importa si es la «izquierda» o la «derecha»); la ALU hace siempre automáticamente referencia a este acumulador como a una de las entradas.
En las operaciones aritméticas y lógicas, por tanto, uno de los operandos estará en el acumulador y el otro se encontrará normalmente en una localización de la memoria.
El resultado será depositado en el acumulador. La referencia al acumulador cómo fuente y destino de los datos es la razón de su nombre: él acumula los resultados.
La ventaja de esta aproximación basada en el acumulador es la constituida por el hecho de que se pueden emplear instrucciones mucho más cortas en código máquina.
Si también el otro operando debiera ser tomado de uno de los otros registros (distintos del acumulador), sería necesario utilizar instrucciones más largas para indicar la dirección del registro.
Por esta razón, la arquitectura del acumulador se resuelve en una mayor velocidad de ejecución.
La desventaja es que el acumulador debe siempre ser cargado con los datos pedidos por la operación antes de su utilización.
Esto puede provocar en algunos casos algunas ineficiencias.

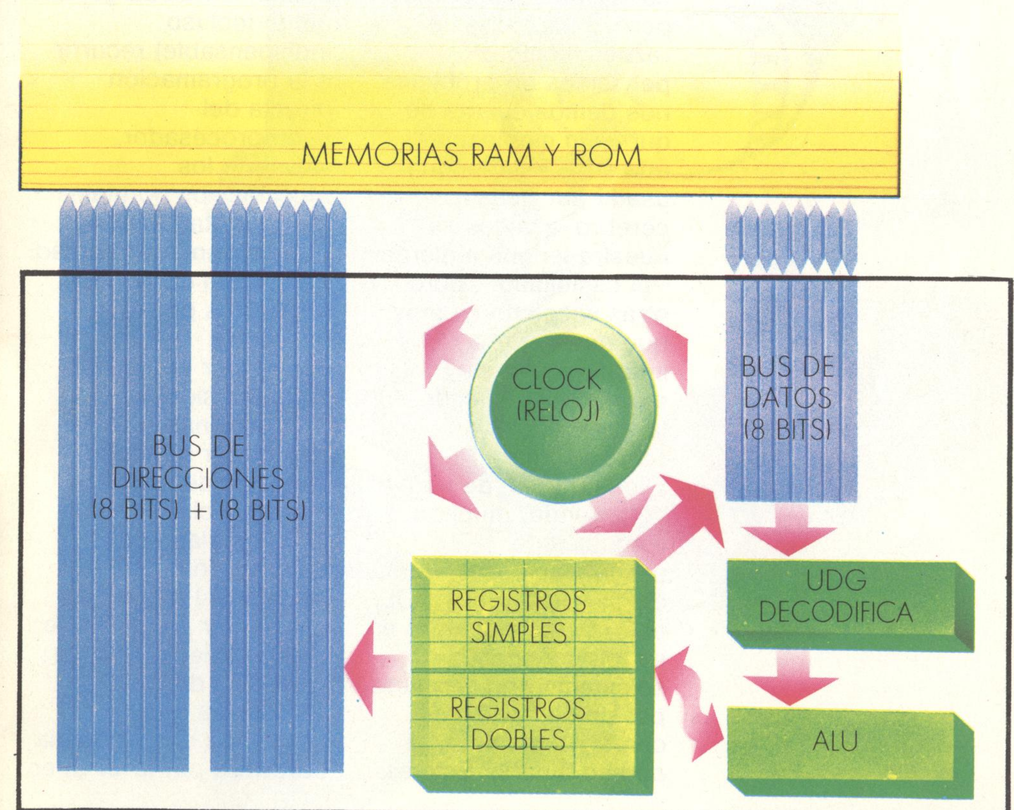


Los buses

El microprocesador se comunica con los periféricos a través de tres buses:
– el bus de direcciones
– el bus de datos
– el bus de control.
El bus de direcciones es un conjunto de líneas eléctricas, que permiten al microprocesador seleccionar una posición de memoria o un registro de un interface. La CPU envía sobre estas líneas, hacia un periférico, una dirección codificada en binario. El periférico, recibida y decodificada la dirección, selecciona el registro correspondiente.
El número de líneas del bus de direcciones determina lo que se llama potencia de direccionamiento del microprocesador; por ejemplo, 16 líneas permiten direccionar 2↑16 (65536) localizaciones de memoria. Es notable —entre otras cosas—que el concepto de dirección es muy similar al que se usa en el lenguaje corriente: la localización de una persona en una ciudad tiene lugar efectivamente a través de una dirección que contiene la calle y el número.
El bus de datos está también constituido por un grupo de líneas eléctricas en las cuales tiene lugar el intercambio de datos entre el microprocesador y los periféricos (memoria e interfaces). El número de líneas de este bus depende de la longitud de palabra del microprocesador: ya que el microprocesador del C64 es de 8

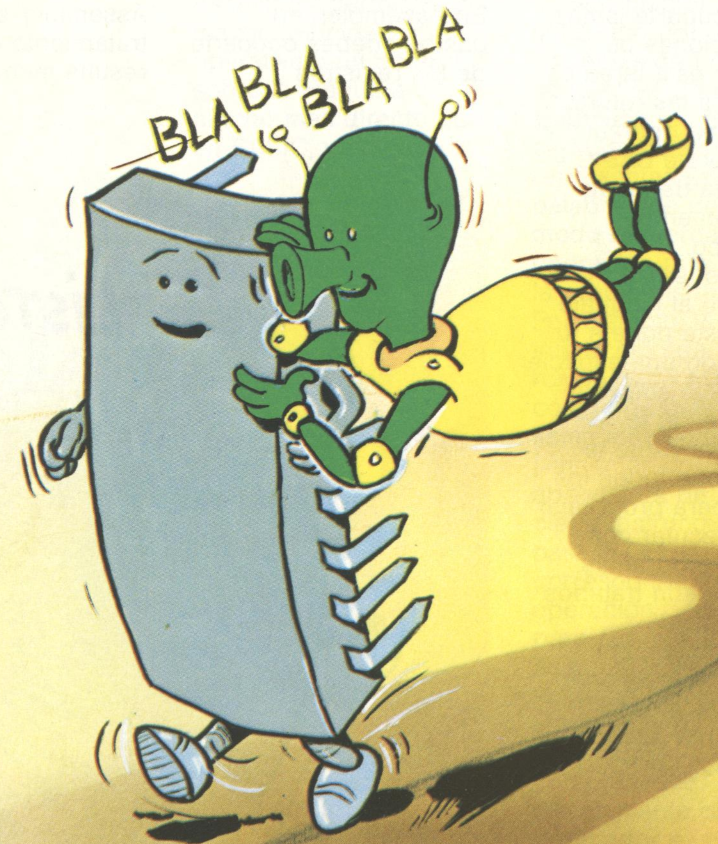
HARDWARE

bits, el bus de datos dispone también de 8 líneas.
El bus de control está constituido por un cierto número de señales de diverso tipo, que aseguran la sincronización entre el microprocesador y los periféricos. Las funciones más comunes garantizadas por este bus son:
– la selección de una operación de lectura o escritura
– la interrupción del microprocesador (por interrupción se entiende un procedimiento que – a través de la señal eléctrica en una patilla especial de la CPU– indica al microprocesador que un periférico quiere comunicarse con él)
– la petición de acceso al bus de un periférico
– el reconocimiento de una petición de acceso al bus
– otras funciones menos comunes, pero tan importantes como las comentadas hasta ahora.

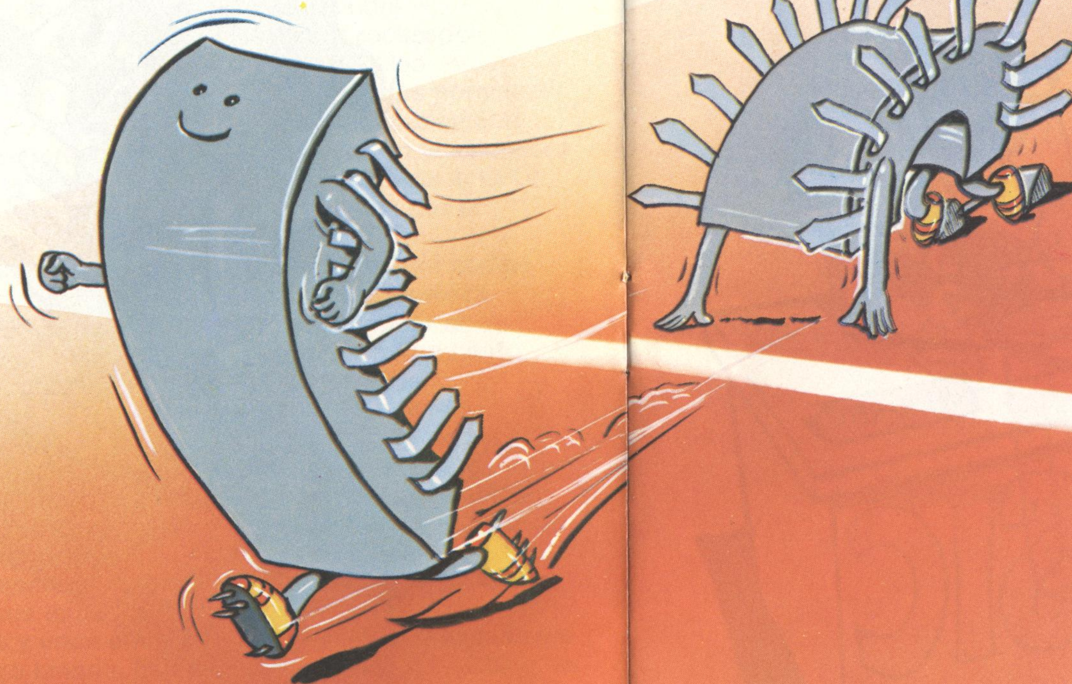


El código máquina

Hemos dicho ya varias veces que escribir un programa significa escribir una cierta serie de instrucciones en un lenguaje comprensible para el ordenador, es decir, en un lenguaje de programación.
Hagamos por un momento una analogía con el hombre.
Nosotros podemos hablar muchas lenguas (inglés, francés, alemán, etc.), podemos comprenderlas todas, pero si hacemos un razonamiento o pensamos un problema, nos damos cuenta de que sólo empleamos una sola. Esta lengua, usada por nuestro cerebro, a veces es nuestra lengua materna —el castellano—, pero otras veces no es más que una «lengua interna». Algo muy parecido les sucede también a los ordenadores.
Ellos pueden, de hecho, comprender muchos lenguajes de programación, el BASIC, el FORTRAN, el COBOL, el PASCAL, etc., pero en su interior usan solamente uno, formado por una larga fila de ceros y unos en código binario. Este es el lenguaje de la máquina. Cualquier instrucción —por ejemplo, en BASIC— debe ser traducida a código máquina para poder ser ejecutada por el ordenador. Esta misión es realizada en el BASIC de tu C64 —ya lo sabes muy bien— por el incansable intérprete BASIC, auténtico traductor simultáneo entre tú y tu ordenador.
En algunos casos puede, sin embargo, ser útil (o incluso indispensable) recurrir a la programación directa del microprocesador, salvando los mecanismos normales (que inevitablemente disminuyen la velocidad de cálculo de la CPU) impuestos por el intérprete.
Programar en código máquina significa proporcionar al ordenador un cierto conjunto de configuraciones binarias (llamadas también códigos operativos) que el ordenador es capaz de comprender y ejecutar; cada una de ellas representa una operación ejecutada vía hardware por la CPU, es decir, mediante las propias conmutaciones de interruptores en el interior del ordenador.
La diferencia fundamental es que para programar en BASIC no es necesario conocer como funciona el microprocesador con el cual se ha realizado el ordenador, mientras que esto sí es necesario para programar en Assembler (así se llama



normalmente el código máquina).
En BASIC, salvo que se usen las instrucciones que leen y escriben directamente en los bytes de memoria (es decir, PEEK y POKE) no debes ocuparte jamás de direcciones de memoria: esta tarea es confiada a las rutinas del sistema operativo y del intérprete BASIC y se efectúa de modo completamente automático.
Una vez que has aprendido el lenguaje BASIC, este pone a tu disposición una especie de interface software con el cual tú te comunicas y que te proporciona todos los medios para programar y para ejecutar con éxito tus programas.
Los datos son tratados con nombres simbólicos (es, por tanto, posible asignar un número a cada nombre: por ejemplo LET PRECIO = 5000); las referencias a puntos específicos del programa (GOTO, GOSUB) se obtienen con referencias a los números indicadores de las líneas del programa BASIC.
En Assembler, en cambio, debes ocuparte de los registros funcionales del microprocesador, de las direcciones de memoria y cada operación debe ser pensada en todos sus detalles, siguiendo las características del ordenador. Además, en Assembler el tratamiento de datos resulta menos sencillo.



LENGUAJE

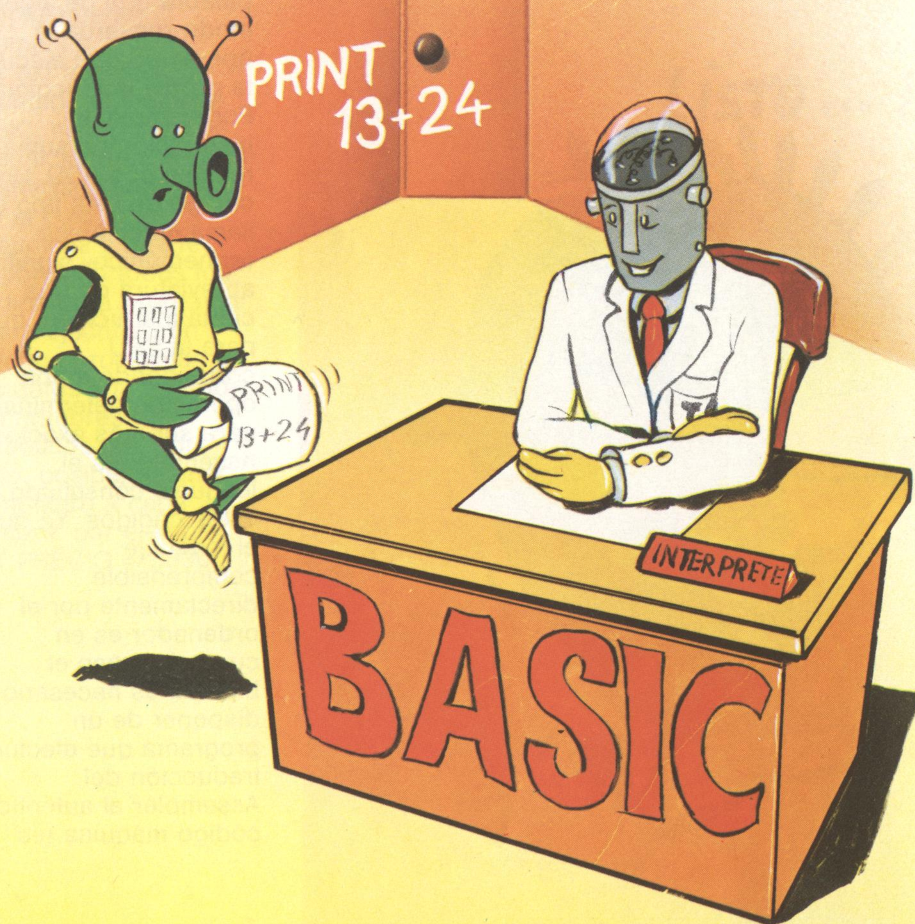
Inconvenientes y ventajas del Assembler

El código máquina está en binario: puedes intuir que escribir un programa en este lenguaje es largo, trabajoso y difícil.
Por otra parte los programas en Assembler permiten una mayor velocidad operativa y un control sobre la máquina mucho más directo que en BASIC, y en ciertos casos es por esto necesario o conveniente someterse a este trabajo, cuando la aplicación lo requiere.
Existen diversas posibilidades de escribir programas en código máquina, utilizando por ejemplo —además de un código numérico— un código simbólico, en el cual cada instrucción está representada por una palabra que de algún modo recuerda la operación que ejecuta la propia instrucción.
Por ejemplo, la instrucción de suma se resume con ADD. Un código de este tipo se llama mnemónico («mnemónico» significa abreviatura de una cierta instrucción, que para la CPU corresponde a una operación determinada); para ser más exactos el Assembler es el lenguaje constituido por estos códigos. Ya que el lenguaje comprensible directamente por el ordenador es en cualquier caso el binario, es necesario disponer de un programa que efectúe la traducción del Assembler al auténtico código máquina (es

decir, que sustituya el código mnemónico de una determinada instrucción por el correspondiente código numérico); este programa recibe el nombre de ensamblador.

Existen muchos ensambladores en el mercado: su uso para quien desee programar seriamente en Assembler, es prácticamente indispensable. Teniendo en cuenta el carácter de introducción al código máquina de esta y de las próximas lecciones, no es de ninguna forma absolutamente necesario que te apresures en procurarte uno.

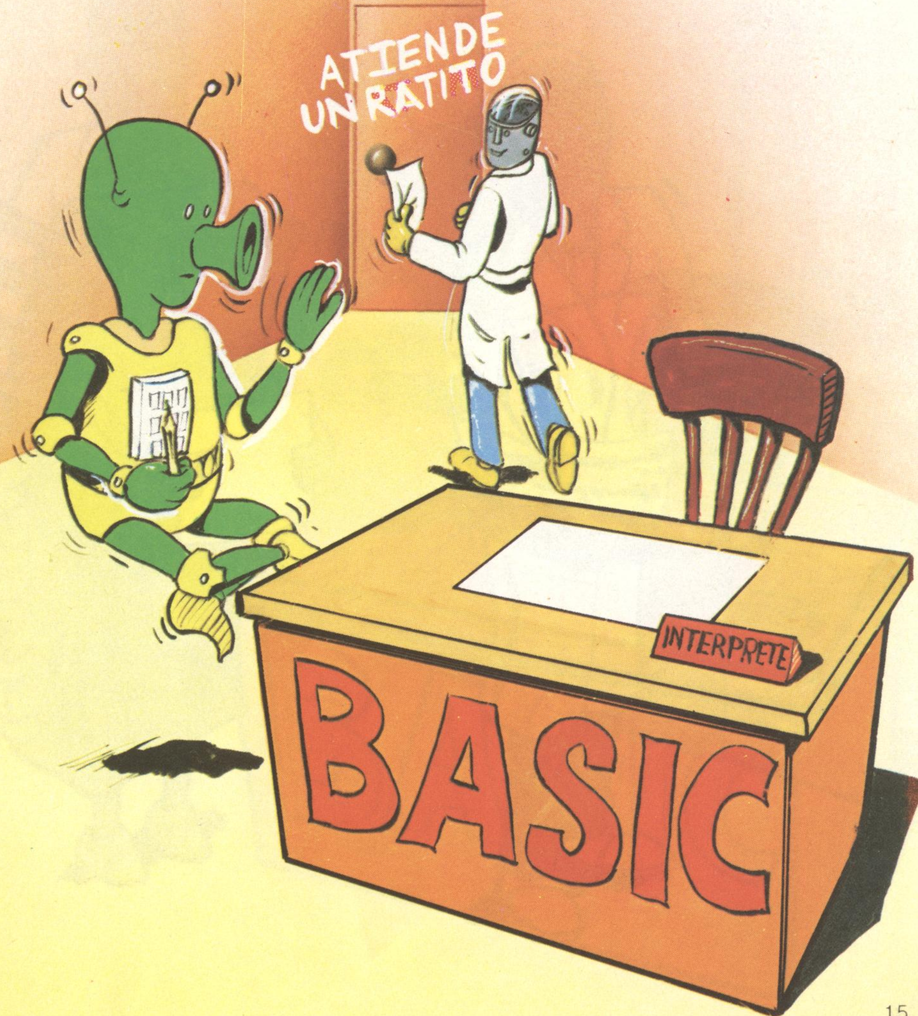
Para terminar,



indicaremos cuáles son las situaciones principales que pueden justificar el empleo del Assembler:

— todas las ocasiones en las cuales el factor tiempo juega un papel fundamental. Entre el tiempo de ejecución de un programa que funciona bajo el intérprete BASIC y el de un programa escrito directamente en Assembler puede existir una diferencia de velocidad enorme (el código máquina puede ser hasta 2-300 veces más rápido);

— todas las ocasiones en las cuales el factor espacio tiene una importancia esencial. La dimensión de un programa BASIC,

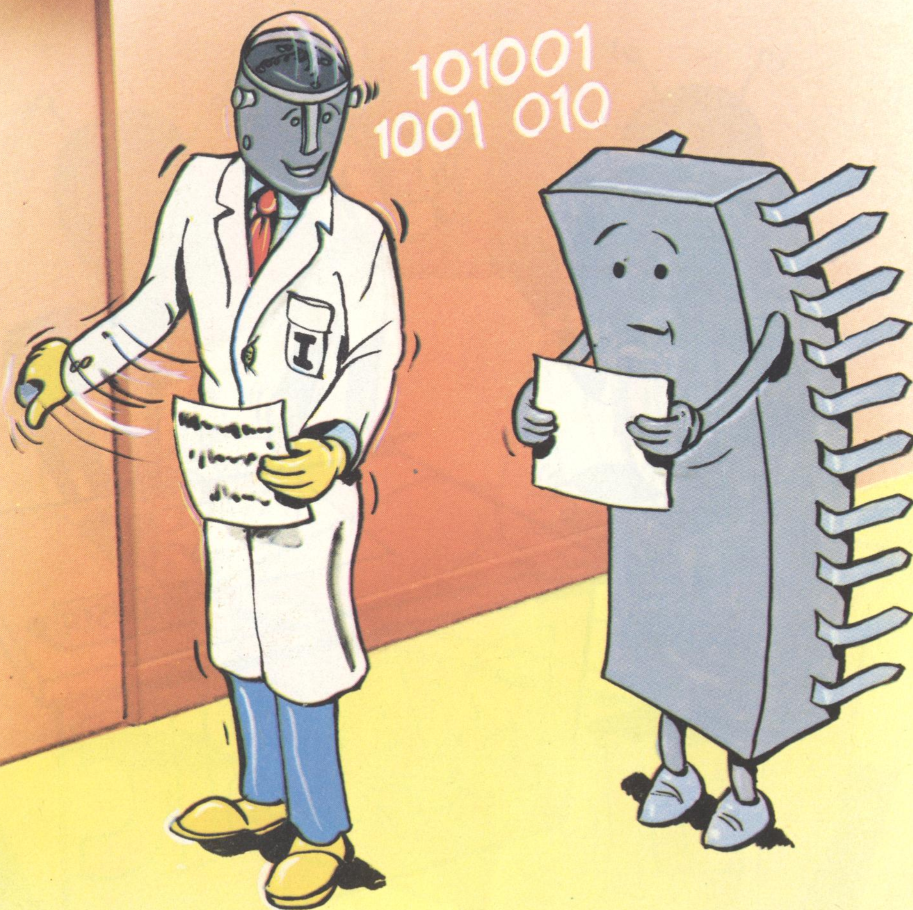


añadida a la de su intérprete, será siempre superior a la del mismo programa (que ejecuta la misma función) realizado en Assembler.

Este factor, con la progresiva e imparable caída de los precios de las memorias, está convirtiéndose en cualquier caso en el menos importante;

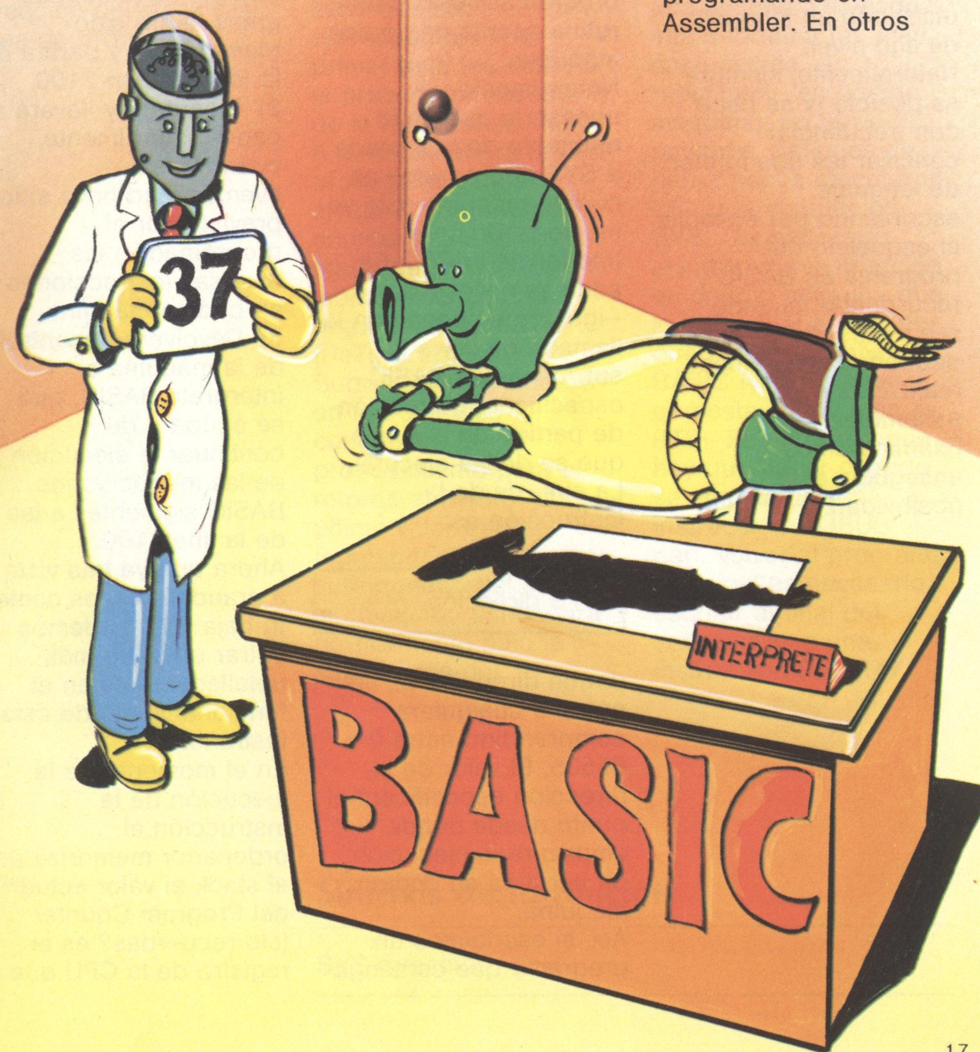
— todas las ocasiones en las cuales es necesario realizar instrucciones que son desconocidas para el intérprete BASIC, o imposibles de realizar con un lenguaje de alto nivel (como por ejemplo el BASIC).

Dado que un programa en Assembler resulta mucho más rápido que un programa en BASIC, pero requiere mucho más tiempo para escribirlo y resulta más



LENGUAJE

difícil encontrar los errores, será siempre necesario tomar de vez en cuando una decisión acerca del camino que conviene elegir, valorando bien las diversas exigencias, las posibles alternativas y los respectivos costes (no sólo en términos de tiempo, sino también en los de trabajo). Además, tienes que tener presente que ciertas aplicaciones particulares, como la utilización del ordenador para juegos de animación, se pueden llevar a cabo solamente programando en Assembler. En otros



casos (ejemplo típico: los programas de gestión financiera o de contabilidad) se requieren en cambio fundamentalmente facilidades de legibilidad y modificabilidad de los programas, obtenibles exclusivamente utilizando un lenguaje de alto nivel.
Naturalmente, siempre es posible (y se hace con frecuencia) conciliar las dos formas de lenguaje, escribiendo por ejemplo el esqueleto del programa en BASIC y recurriendo, cuando sea necesario, a pequeños programas realizados en Assembler.
Examinaremos enseguida esta posibilidad.

SYS

SYS es un comando BASIC abreviatura de la palabra «System».
Resulta muy útil cuando se desea pasar directamente de un programa BASIC a una rutina escrita en código máquina.
Naturalmente, es necesario que en el momento de la llamada a SYS el programa en código máquina esté ya memorizado en el interior de tu C64. Es además necesario —igual que sucede en la llamada de las subrutinas en BASIC— especificar la dirección de partida de la rutina que se desea ejecutar.
La sintaxis de la instrucción es:

```
SYS dirección
```

donde dirección es un número cualquiera comprendido entre 0 y 65535. El valor de la dirección especificará el punto desde donde empezará la ejecución de la rutina en código máquina.
Así, si escribimos un programa que contenga una línea BASIC del tipo:

```
100 SYS 2100
```

nuestro C64 ejecutará en orden esta serie de operaciones:
1) Pasará el control del BASIC a la rutina en código máquina, que antes habrá sido memorizada y partirá de la localización 2100.
2) Ejecutará y llevará a cabo (naturalmente, cuando esta eventualidad haya sido prevista por el programador) las diversas instrucciones en código máquina.
3) Devolverá el control de la máquina al intérprete BASIC, que se ocupará de continuar la ejecución de las instrucciones BASIC siguientes a las de la línea 100.
Ahora que ya has visto a grandes rasgos como trabaja SYS podemos entrar un poco más detalladamente en el funcionamiento de esta instrucción.
En el momento de la ejecución de la instrucción el ordenador memoriza en el stack el valor actual del Program Counter (¿lo recuerdas? es el registro de la CPU que

le recuerda al microprocesador las direcciones de las distintas instrucciones que le faltan por realizar): de esta forma, al final de la rutina en código máquina el intérprete BASIC sabrá encontrar —tomando su dirección del stack— la línea de programa desde donde debe continuar la ejecución.
Por lo tanto, en el Program Counter se coloca la dirección especificada en SYS, y desde allí arrancará el programa en código máquina. Es obvio que el valor de la dirección tiene que ser el correspondiente al primer byte (es decir, a la primera intrucción) de la rutina a ejecutar.
A partir de ahora la CPU proseguirá con la ejecución del programa en código máquina; si en un determinado momento encontrara una instrucción RTS (ReTurn from Subroutine), se pondrá en marcha y se completará el procedimiento para el retorno al BASIC: el último valor del PC (contador de programa) será quitado del stack y la ejecución continuará normalmente bajo la supervisión del intérprete.
Aunque los ejemplos que examinaremos más adelante (en la parte de la lección dedicada a la programación) esclarecerán estos hechos, es importante recordar que con SYS el ordenador ejecuta una auténtica llamada a una subrutina (en este caso escrita —lo repetimos— en código máquina). En otras palabras, después de haber hecho ejecutar al C64 una instrucción SYS... es necesario insertar una instrucción en código máquina que le indique al ordenador que tiene que volver al BASIC (ya hemos visto que esta instrucción es RTS). Es lo mismo que hay que hacer cuando se emplea una instrucción GOSUB, para volver al programa principal es necesario escribir al final del subprograma una instrucción RETURN.

Sintaxis de la función

SYS dirección

LENGUAJE

USR

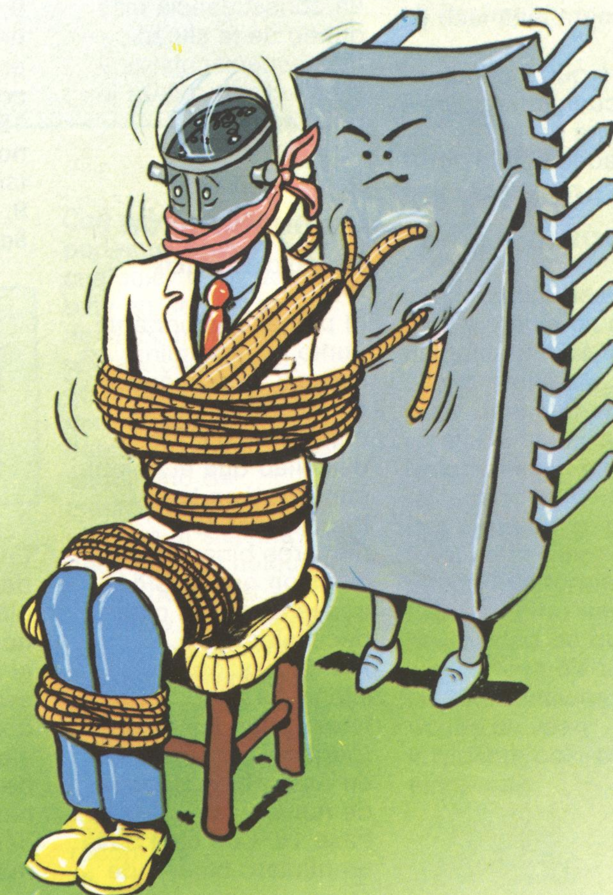
USR (abreviatura de User SubRoutine) es otra instrucción –para ser más exactos es una función– utilizable para realizar desde el BASIC llamadas a partes de programas escritas en código máquina.
Realiza aproximadamente el mismo trabajo de SYS, pero lo hace de una forma ligeramente más complicada. Por esta razón, USR es de empleo menos frecuente que SYS.
La sintaxis de esta función es

```
A = USR (C)
```

y para el ordenador significa: «transfiere el valor de la variable C al programa en código máquina cuya dirección de salida está memorizada en las localizaciones de memoria número 1 y 2. A la salida de la rutina asignarle el resultado a la variable A».
Hablando en términos un poco más riguroso (pero seguramente más exactos) USR sirve para ejecutar un programa en código máquina cuya dirección inicial debe haber sido previamente memorizada en los bytes 1 y 2 mediante instrucciones POKE. El valor del argumento de USR se memoriza en una parte del ordenador llamada «acumulador de coma flotante», que empieza en el byte 61; en este acumulador (comparable a una celda de memoria) se encontrará después el resultado del cálculo realizado. De cualquier forma este resultado también está disponible para el programa BASIC como valor de la función.
También en este caso la rutina en código máquina tiene que terminar con la instrucción RTS.
La ventaja de USR con respecto a SYS es que el mismo programa se puede usar para trabajar distintas veces con distintos valores de entrada.
La desventaja, en cambio, la constituye el hecho de que sólo es posible pasar a la rutina un valor a la vez.
Además, empleando más de una rutina será necesario –antes de acceder a ella– insertar mediante POKEs en las localizaciones 1 y 2 las direcciones de salida del programa. En los casos en los que el factor velocidad sea esencial, este hecho puede resultar extremadamente poco ventajoso, puesto que requiere en suma una inútil pérdida de tiempo.

Sintaxis de la función

USR (argumento)



Descubriendo el Assembler

Antes de pasar directamente a la escritura de programas en código máquina, debemos todavía afrontar otros temas importantes (es más, imprescindibles), gracias a los cuales podremos avanzar de una manera más fácil y sencilla.
Te podrá quizá parecer que el código máquina no es para tí: demasiadas cosas que aprender, que saber y que tener en la cabeza. Esto también puede ser cierto: sin embargo, una vez aprendidos los conceptos fundamentales, muchos temas te resultarán mucho menos complicados de lo que podría parecer a primera vista.
Además, cuanto más te adentras en el assembler —haciéndote en consecuencia más dueño de la situación— más espectaculares y satisfactorios serán los resultados.

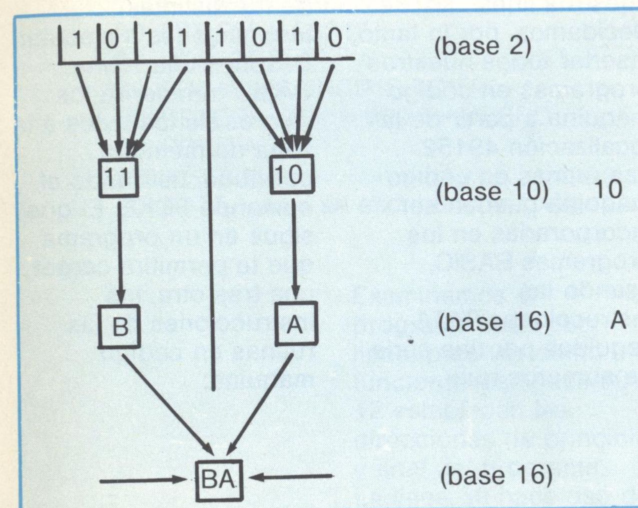
La numeración hexadecimal

El primer e importante punto es determinar un sistema de numeración que llegue a un compromiso entre la dificultad que el hombre encuentra en leer y trabajar con los números binarios y la aversión que la CPU posee hacia las cifras decimales.
Por esta razón, interviene un tercer y fundamental (para quien programa en Assembler) sistema de numeración, el de base 16. Con esta base un número binario de ocho cifras puede ser escrito con dos cifras en base dieciséis, es decir, con dos números hexadecimales.
Seguramente recordarás que un número en base dos está compuesto solamente por las cifras 0 y 1, mientras que uno en base 10 se escribe con cifras comprendidas entre 0 y 9; así, un número en base 16 deberá ser escrito con cifras comprendidas entre 0 y 15. Sin embargo, como no tenemos cifras (simples) mayores que 9, usamos las primeras seis letras del alfabeto:

10 = A
11 = B
12 = C
13 = D
14 = E
15 = F

El modo más sencillo de convertir un número binario de 8 bits en un número hexadecimal es el que consiste en separar los 8 bits en dos grupos de 4 bits, efectuar la conversión de cada grupo de una base a la otra y después combinar el resultado.

Por ejemplo:



Con este método podemos convertir cualquier valor de un byte en dos cifras hexadecimales. La ventaja consiste en obtener una escritura más compacta de la notación binaria y, con un mínimo de entrenamiento, casi idéntica al sistema decimal. En cualquier caso, todos estos sistemas de numeración son —recordémoslo— modos diferentes de representar los mismos números. El único secreto es conocer la base utilizada. Por convención, a fin de reconocer inmediatamente un número haxadecimal, escribiremos siempre los números en base 16 con el sufijo $. Así

10

significará «diez decimal», mientras que

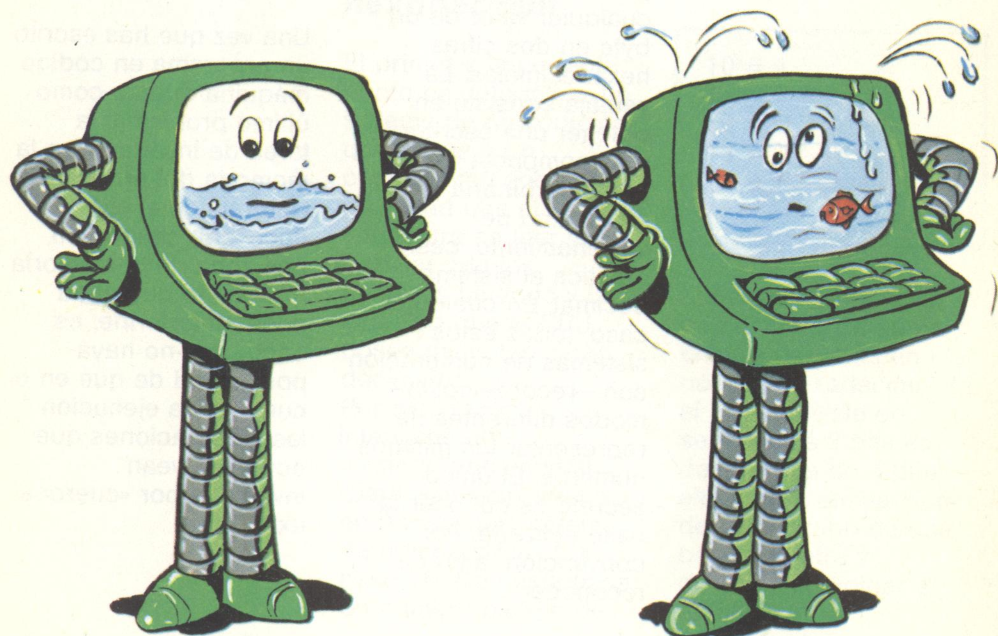
$ 10

significará «diez hexadecimal» (es decir, 16 decimal).

Cómo memorizar los programas en código máquina

Una vez que has escrito un programa en código máquina queda, como último problema, la tarea de insertarlo en la memoria del ordenador. Naturalmente, es necesario situarlo en una zona de la memoria RAM en la que nada pueda interferirle, es decir, que no haya posibilidad de que en el curso de la ejecución las localizaciones que ocupa se vean invadidas por «cuerpos extraños».

Un excelente lugar, por ejemplo, es el que arranca en la localización 49152 y llega hasta la 53247: en este área de memoria existe mucho espacio inutilizado, que parece hecho a posta para nuestros fines.
Decidamos, por lo tanto, insertar todos nuestros programas en código máquina a partir de la localización 49152.
Las rutinas en código máquina pueden ser incorporadas en los programas BASIC, usando las instrucciones DATA seguidas por una serie de números que representan los códigos de las distintas instrucciones a ejecutar. Después una rutina BASIC transferirá los valores de los bytes a la zona de memoria oportuna, utilizando el comando POKE. El que sigue en un programa que te permitirá cargar, una tras otra, las instrucciones de las rutinas en código máquina:



PROGRAMACION

```
10 LET PRINCIPIO = 49152
20 LET FIN = 53247
30 RESTORE
40 FOR A = PRINCIPIO TO FIN
50 READ X : IF X = 999 THEN END
60 POKE A, X
70 NEXT A
80 DATA... aquí se escriben los códigos...
```

Examinemos el programa línea por línea, para ver cómo funciona: las líneas 10 y 12 establecen las direcciones de principio y final del programa.
La línea 30 hace uso de la instrucción RESTORE para posicionar el puntero de los DATA sobre el primero de los datos que tendrán que ser leídos.
La línea 40 empieza un bucle que va desde la localización PRINCIPIO hasta la localización FIN. La línea 50 lee los diversos elementos de las líneas DATA y comprueba en cada caso que no haya uno igual a 999 (que es un código no admitido para la instrucción POKE); de esta manera somos capaces de identificar el final de los datos. La línea 70 cierra el bucle.
Naturalmente, no es necesario disponer todos los valores en una única línea DATA; puedes usar cuantas líneas desees, siempre que sean compatibles con la memoria disponible. La línea 80 (y las eventualmente siguientes) contiene los valores decimales de la rutina y debe tener como último elemento el 999, para que el ordenador pueda darse cuenta que ha llegado hasta el final.
Así este programa carga el código máquina en la RAM.
Para hacer ejecutar la rutina será necesario modificar el programa, de tal forma que al final éste ejecute la rutina, incluyendo, por lo tanto, una línea SYS o USR.
Este ejercicio (en conjunto verdaderamente elemental) lo dejamos a tu cargo.
Como alternativa, puedes ordenar la ejecución impartiendo uno de los dos comandos en modo inmediato.

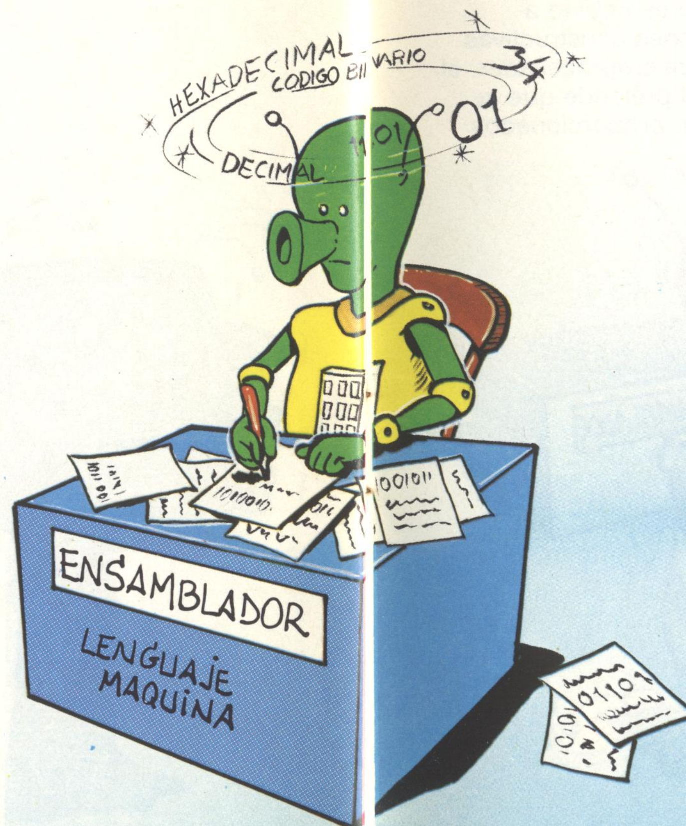
Ejemplos Assembler

Probemos ahora a escribir algunos programas muy sencillos en Assembler, mostrándote simultáneamente su funcionamiento y su comparación con el correspondiente listado BASIC. No siempre es posible hacer esta comparación: los ejemplos que trataremos ofrecerán, sin embargo, esta posibilidad.
Como primer ejemplo propongámonos transferir el contenido de dos localizaciones de memoria (por ejemplo, la 58000 y la 58001) a otra pareja de localizaciones (elegimos la 34567 y la 34568).
En BASIC escribiríamos:

```
10 POKE 34567, PEEK (58000)
20 POKE 34568, PEEK (58001)
```

y todo estaría resuelto. Veamos ahora qué hay que hacer en Assembler. He aquí el listado:

```
LDA $E290
STA $8707
LDA $E291
STA $8708
RTS
```

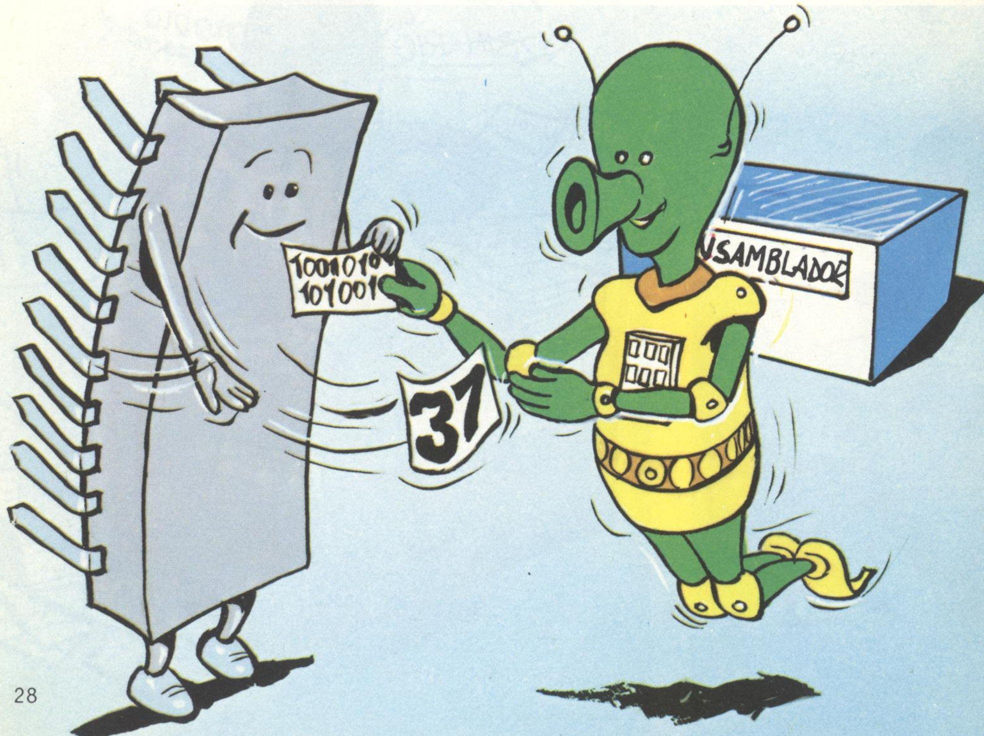


La primera línea carga en el acumulador (LDA es la abreviatura de LoaD Accumulator, es decir, carga el acumulador) el valor contenido en la localización 58000 (E290 hexadecimal). La segunda línea memoriza, en cambio, este valor (STA significa STore Accumulator, memoriza el acumulador) en las localizaciones 34567 (8707 hexadecimal). La misma tarea pero realizándola sobre las localizaciones 58001 y 34568 se lleva a cabo con las dos líneas siguientes. La última línea es el famoso RTS (Return from SubRoutine) que sirve para devolverle el control al BASIC.
Observemos ahora la constitución de las instrucciones: un operador y uno o más operandos. En LDA $E290 –por ejemplo– LDA es el operador, mientras que $E290 es el operando.
Es necesario hacer una precisión sobre la parte, (o, mejor dicho, sobre el campo) relativa a los operandos.

Este campo depende estrechamente del tipo de operación, y pertenece siempre a o uno de los siguientes grupos:
– de un sólo argumento
– de dos argumentos, separados por un carácter especial (generalmente una coma).
Pasemos ahora a la conversión del programa en los códigos numéricos (estos que acabamos de ver son solamente los mnemónicos). Es necesario tener en las manos la tabla proporcionada por el fabricante del microprocesador, y convertir uno a uno los diversos términos. A cada elemento le corresponderá naturalmente un código numérico preciso:

LDA $E290	AD 90 E2
STA $8707	8D 07 87
LDA $E291	AD 91 E2
STA $8708	8D 08 87
RTS	60

Los números que hemos escrito están todavía en hexadecimal: antes de ser escritos en los DATA deberán ser convertidos en decimales.
La última cosa a tener en cuenta es que –si miras bien– las direcciones de las localizaciones han sido escritas invertidas (es decir, E290 ha sido escrito como 90E2, e igualmente ha ocurrido con los demás).
Esto es debido a razones constructivas del microprocesador, el cual pretende que se sean proporcionados



antes el byte bajo y depués el alto de cualquier dirección.
Al final obtenemos los valores siguientes:

$AD	173
$90	144
$E2	226
$8D	141
$07	7
$87	135
$AD	173
$91	145
$E2	226
$8D	141
$08	8
$87	135
$60	96

que, insertados en la línea DATA (acordándose de añadir también el 999), formarán la rutina.
Ejecuta el programa y envía tanto un SYS para poner en marcha el código máquina.
Intenta después comprobar con PEEK en las localizaciones 34567, 34568, 58000, 58001 que el trabajo haya sido verdaderamente llevado a cabo.
Probemos ahora este segundo ejemplo:
Deseamos situar en la lozalización de memoria número 20000 el valor 19. Deseamos además que el número se duplique y se coloque en la localización 20001. En BASIC escribiríamos.

```
10 POKE 20000, 19
20 POKE 20001, PEEK (20000) * 2
```

y el intérprete se ocuparía de las diferentes operaciones.
En Assembler, en cambio, nosotros nos tenemos que ocupar de todo.
He aquí el listado:

```
LDA $13
STA $4E20
CLC
ROL A
STA $4E21
RTS
```

Intentemos comprender qué es lo que se ha hecho: la primera línea carga en el acumulador el valor hexadecimal 13 (19 decimal), mientras que la segunda lo memoriza en la localización $4E20 (20000 decimal). A a continuación se prepara la multiplicación. Puesto que en binario multiplicar por 2 significa desplazar una posición hacia la izquierda cada bit (por ejemplo, 100 binario –igual a 4– se convierte en 1000 –igual a 8–), bastará con ejecutar la llamada operación de rotación (mnemónico ROL).

Dado que el bit que se añade por la derecha se toma del carry (que es un bit especial, que forma parte del registro de estado del microprocesador), será necesario ponerlo a cero, para evitar que influya erróneamente en el resultado: CLC (Clean the Carry, pon a cero el carry) sirve precisamente para esto. Finalmente el resultado es memorizado en $4E21 (20001 decimal). La conversión del programa a sus códigos numéricos lleva a los siguientes valores:

169, 19, 141, 32, 78, 24, 42, 141, 33, 78, 96, 999

Si ejecutas esta rutina podrás comprobar cómo los dos programas –en BASIC y en codigo máquina– desempeñan el mismo trabajo y llegan al mismo resultado. Profundizaremos la disertación sobre el código máquina en las siguientes lecciones.

Código máquina para imprimir donde gustes

He aquí un ejemplo de cómo usar en un programa en BASIC rutinas en código máquina ya existentes en la ROM del BASIC o del sistema operativo. En el caso a examen nos valemos de una llamada a la localización 58732 en la que empieza una rutina capaz de poner al día la posición de impresión en pantalla en base a los datos leídos en las localizaciones 211 (X) y 214 (Y).
Es una forma eficaz de obtener una instrucción del tipo: «PRINT AT», que no existe en las instrucciones estándar del CBM BASIC.

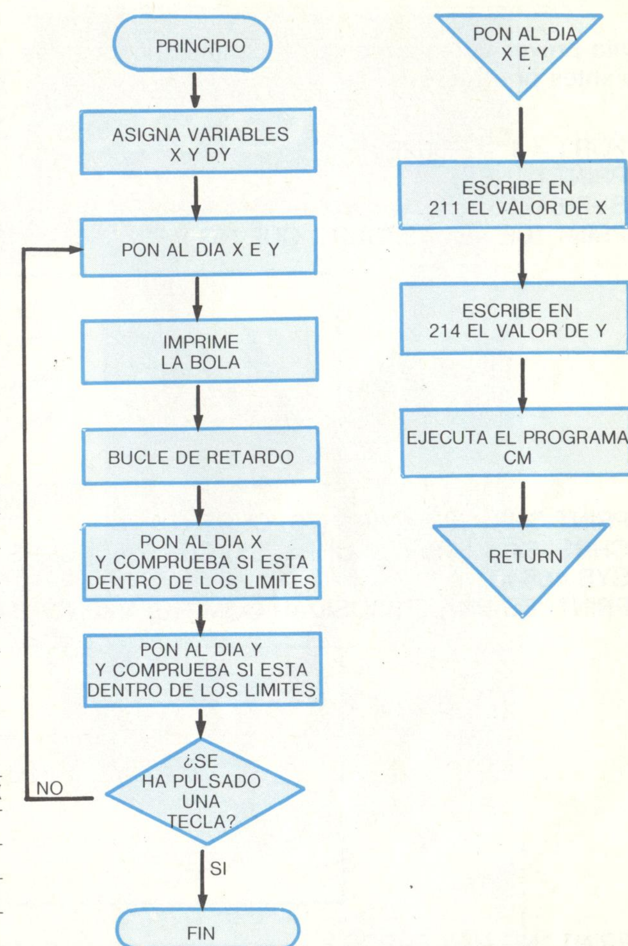
Este es el desensamblado de las rutinas interesadas tras la llamada con SYS 58732.

E56C	A6 D6	LDX $D6
E56E	A5 D3	LDA $D3
E570	B4 D9	LDY $D9, X
E572	30 08	BMI $E57C
E574	18	CLC
E575	69 28	ADC #$28
E577	85 D3	STA $D3
E579	CA	DEX
E57A	10 F4	BPL $E570
E57C	20 F0 E9	JSR $E9F0
E57F	A9 27	LDA #$27
E581	E8	INX
E582	B4 D9	LDY $D9, X
E584	30 06	BMI $E58C
E586	18	CLC
E587	69 28	ADC #$28
E589	E8	INX
E58A	10 F6	BPL $E582
E58C	85 D5	STA $D5
E58E	4C 24 EA	JMP $EA24

E570	B4 D9	LDY $D9, X
E572	30 08	BMI $E57C
E574	18	CLC
E575	69 28	ADC #$28
E577	85 D3	STA $D3
E579	CA	DEX
E57A	10 F4	BPL $E570
E57C	20 F0 E9	JSR $E9F0
E57F	A9 27	LDA #$27
E581	E8	INX
E582	B4 D9	LDY $D9, X
E584	30 06	BMI $E58C
E586	18	CLC
E587	69 28	ADC #$28
E589	E8	INX
E58A	10 F6	BPL $E582
E58C	85 D5	STA $D5
E58E	4C 24 EA	JMP $EA24

E9F0	BD F0 EC	LDA $ECF0, X
E9F3	85 D1	STA $D1
E9F5	B5 D9	LDA $D9, X
E9F7	29 03	AND #$03
E9F9	0D 88 02	ORA $0288
E9FC	85 D2	STA $D2
E9FE	60	RTS

EA24	A5 D1	LDA $D1
EA26	85 F3	STA $F3
EA28	A5 D2	LDA $D2
EA2A	29 03	AND #$03
EA2C	09 D8	ORA #$D8
EA2E	85 F4	STA $F4
EA30	60	RTS




```
 5 X = 1 : Y = 4 : DX = 1 : DY = 1
10 GOSUB 45 : PRINT "•"
1& FOR T = 1 TO 10 : NEXT
20 GOSUB 45 : PRINT "  "
25 X = X + DX : IF X = 0 OR X = 24 THEN DX
   = – DX
30 Y = Y + DY : IF Y = 0 OR Y = 23 THEN DY
   = – DY
35 GET R$ : IF R$ = "  " THEN 10
40 END
45 POKE 211, X : POKE 214, Y : SYS 58732
50 RETURN
```

EJERCICIOS

Intenta prever qué aparecerá en pantalla después de la ejecución de los siguientes programas:

```
10 FOR I = 1 TO .002E3
20 PRINT I : NEXT
30 SYS 64738
40 PRINT "DE AQUI SEGURO QUE NO PASA"
```

```
10 PRINT CHR$ (83) CHR$ (79) CHR$ (70)
   CHR$ (84) CHR$ (73) CHR$ (68) CHR$ (69) CHR$ (65)
20 SYS 45640
30 PRINT "ES UNA CURIOSIDAD COMPLETAMENTE IMPREVISIBLE"
```

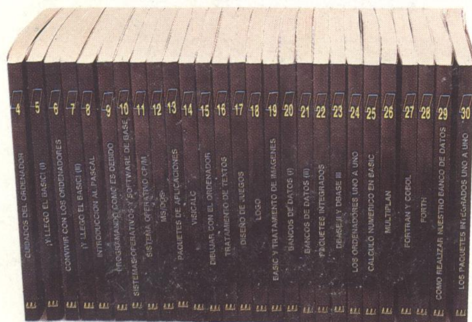
```
10 PRINT "NO HAY ERROR"
20 PRINT "PERO... ¿POR QUE DA ERROR?"
30 SYS 44799
40 PRINT "LA CULPA ES DE LA LINEA 30"
```


¡YA ESTA A LA VENTA!

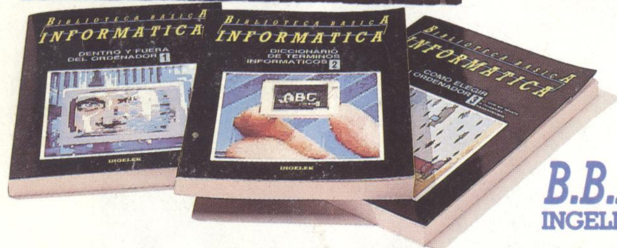
la revista que esperaba el usuario de COMMODORE

novedad en España: «fuera errores»

UNA GRAN OBRA A SU ALCANCE



UNA OBRA COMPLETISIMA EN 30 VOLUMENES QUE TRATA TODOS LOS TEMAS , DESDE QUE ES UN ORDENADOR HASTA EL ESTUDIO DE LOS DIVERSOS LENGUAJES, PASANDO POR LOS LENGUAJES, METODOS DE PROGRAMACION, ELECCION DEL ORDENADOR ADECUADO, DICCIONARIO, ETC.



B.B.I.
INGELEK

30 EXTRAORDINARIOS VOLUMENES DE APARICION SEMANAL CON TODOS LOS CONCEPTOS DE LA INFORMATICA

GRAN OFERTA DE SUSCRIPCION
9.995 PTAS

AHORRE MAS DE 1.000 PTAS Y LLEVESE UNA MAGNIFICA CALCULADORA SOLAR VALORADA EN 2.500 PTAS.

OFERTA VALIDA UNICAMENTE PARA ESPAÑA

SUSCRIBASE POR TELEFONO

Todos los días, excepto sábados y festivos, de 8 a 6,30 atenderemos sus consultas en el

2505820