46
185 pts.
IVA INCLUIDO

RUN

Enciclopedia Práctica del Spectrum

Nueva Lente/Ingelek

FORTH

ACTUALMENTE el lenguaje informático más popular entre los microordenadores es el BASIC, debido a la poca memoria que ocupa y a su relativa facilidad de aprenderlo. Sin embargo, presenta el inconveniente de que al ser «interpretado» se hace excesivamente lento para algunas aplicaciones, como los juegos. Se solucionaría el problema utilizando entonces un lenguaje compilado, pero son más difíciles de dominar y lentos para realizar continuas transformaciones.

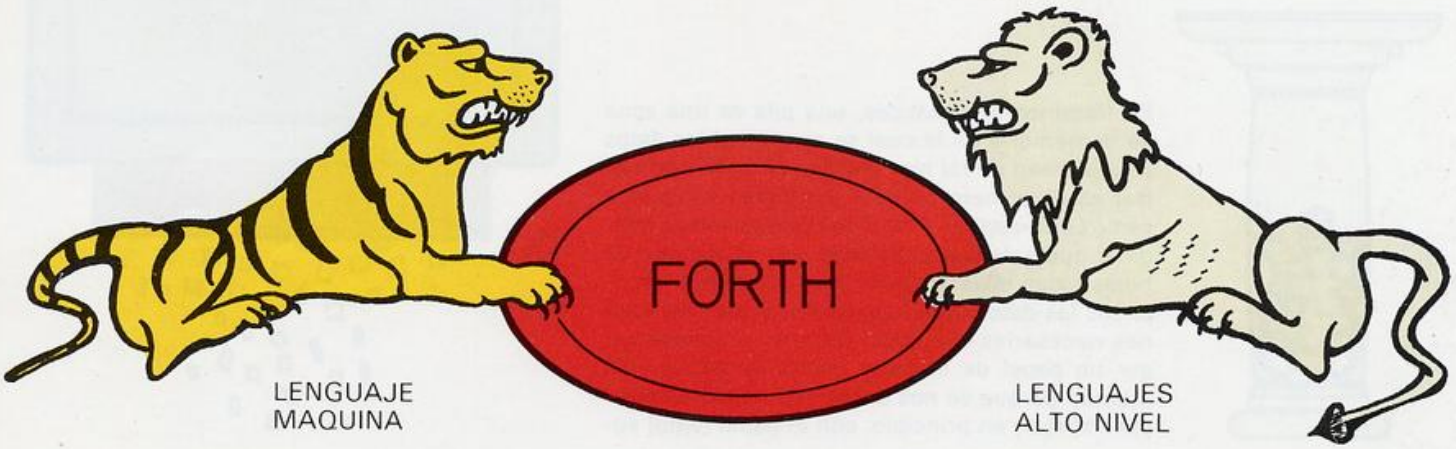
Es evidente que la solución ideal se encuentra en reunir las ventajas de ambos. Y esto es lo que ha conseguido el FORTH: un intérprete muy rápido. Este asombroso prodigio se consigue porque el FORTH trabaja de forma similar a como le gusta al ordenador, cercano al código máquina. Así, por ejemplo, la forma en que guarda la información es en una pila o *stack* de la cual se extraen los datos para operar. Nos podemos imaginar que utiliza una columna en la cual se van apilando los números y los extraemos en orden inverso al de su entrada, como más tarde veremos.

Sin embargo, no todo puede ser perfecto. El FORTH presenta la particularidad o inconveniente de que no puede trabajar con decimales ni «sabe» realizar más operaciones matemáticas que las cuatro básicas (suma, resta, multiplicación y división), lo cual le hace inútil para una gran cantidad de aplicaciones. Entonces ¿cuál es la utilidad de este lenguaje? Sencillamente los juegos (para llevar un marcador no hace falta decimales). ¿Un lenguaje casi diez veces más rápido que el BASIC, intérprete y de alto nivel para hacer juegos? No está mal, ¿verdad?

La ventaja que aporta el FORTH con respecto al BASIC es la de su velocidad, pero su forma de trabajo es complicada.

¿Por qué no conocemos ya a este lenguaje? Con el SPECTRUM podemos, ya que disponemos de al menos dos versiones FORTH; la de *ARTIC*, co-

Su alta velocidad, a pesar de ser intérprete es debido a que se encuentra cercano al lenguaje máquina, pero sin abandonar las características de los lenguajes de alto nivel.

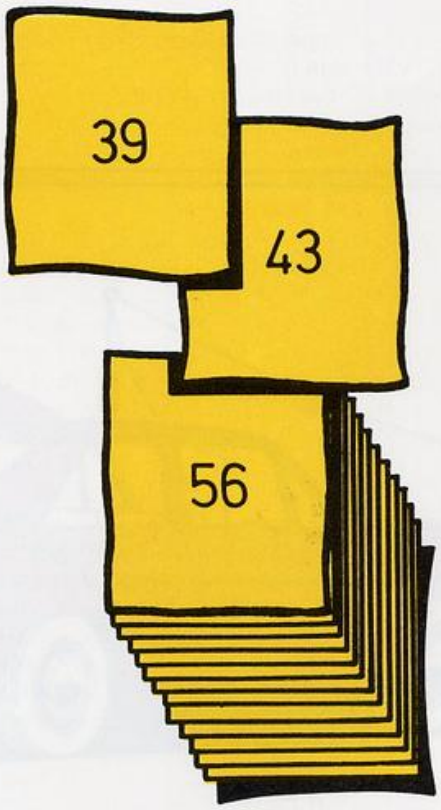


¡!

La única posibilidad que tiene el FORTH de operar es la de tomar los datos de la pila, si no es así no puede realizar operaciones ni funciones de ningún tipo.

*

Una pila o *stack* es una zona de la memoria en la cual se acumulan los datos y se extraen de tal manera que el último en entrar es el primero en salir.

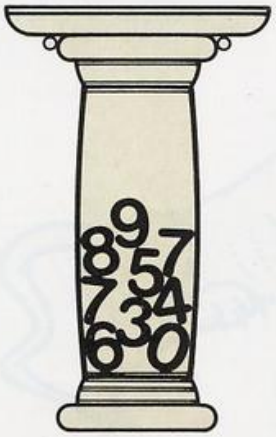


La pila la podemos comparar a unas hojas de papel amontonadas de las cuales sólo podemos trabajar con las de arriba para que así no se nos caigan las hojas.

mercializada por INDESCOMP, y la de *ABERSOFT.*

PILAS O *STACKS*

El FORTH guarda los datos en una columna que se conoce como pila o stack.



En términos informáticos, una pila es una zona de la memoria en la cual se acumulan los datos y se extraen de tal manera que el último en entrar es el primero en salir (en inglés se conoce como *LIFO «Last In, First Out»).* Nos podemos imaginar que estamos trabajando con un montón de hojas todas ellas apiladas y en las cuales escribimos los datos para luego realizar las operaciones necesarias con ellos. No nos es posible coger un papel de la parte central o inferior del montón porque se nos caería todo. Sólo es posible trabajar, en principio, con el papel (dato) superior, o, como mucho, con los tres superiores ya que con éstos no hay riesgo de accidente. Y ¿ésto que tiene que ver con FORTH? Pues nada más ni menos que ésta es la manera en la que trabaja este lenguaje.

Esto nos obliga a que los datos con los cuales vamos a trabajar estén dentro del *stack,* es decir, para operar debemos tener nuestros papeles con los datos escritos en nuestro montón. Veamos un ejemplo:

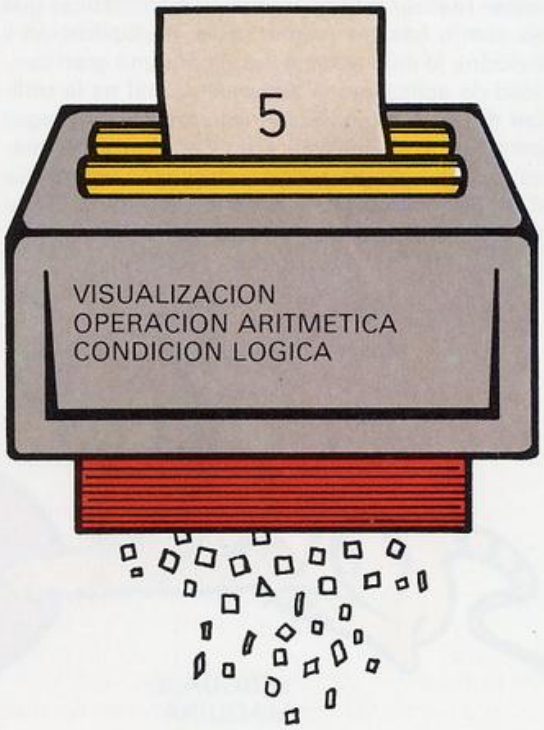
3.

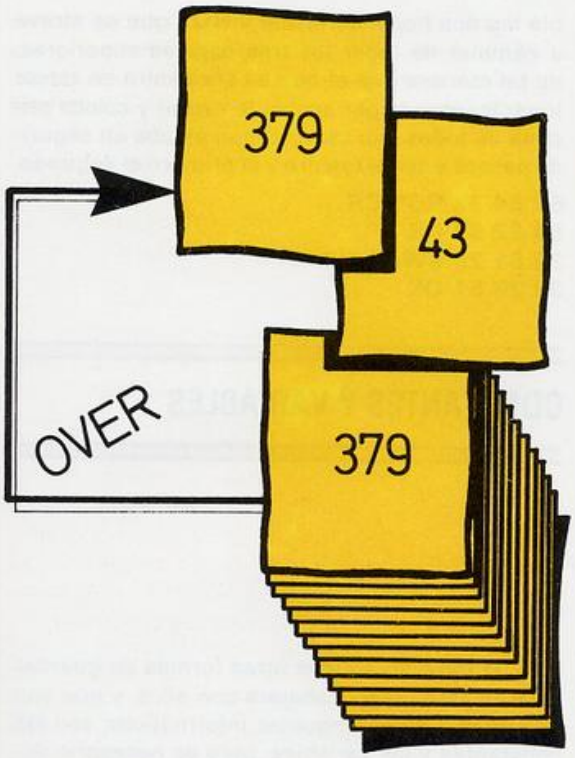
Sencillo ¿verdad? Vamos a ver lo que hace. En primer lugar introducimos un número en la pila (el 3) y después hacemos que se imprima el último número de la pila, que, evidentemente será el 3, gracias al comando "." (en FORTH a los comandos se les llama palabras). Siempre después de realizada una operación con un número de la pila (en este caso la visualización) éste es eliminado de ella. En nuestro ejemplo con el montón de papeles es como si después de ver el número del papel superior le rompiésemos, con lo cual tenemos acceso al siguiente inferior.

456 238 CR.
238 OK

Si hemos entendido el anterior ejemplo, éste nos costará poco. Tras introducir dos números en la pila visualizamos uno de ellos en la línea siguiente (para ello tenemos **CR**). Como el último intro-

Después que se realiza una operación con el número superior de la pila éste es destruido.





OVER *repetirá el valor del penúltimo dato del stack y lo pondrá encima de todos.*

ducido es el 238, éste será el que aparezca. En nuestro ejemplo de los papeles, primero escribimos uno con el 456 y encima ponemos otro con el 238. A la hora de imprimir será éste el que aparezca en primer lugar. Si ahora hacemos:

CR.

Es lógico que aparezca:

456 OK

Al imprimir, y por tanto destruir, el papel con el 238 se quedó al descubierto el siguiente inferior (el 456) que es el que ahora aparece por pantalla. Es decir, los últimos datos en entrar son los primeros en salir.

Hasta aquí todo va bien, pero ¿dónde están las operaciones? Ya mencionamos antes que FORTH opera con los números de la pila, concretamente con los dos últimos, y ambos son sustituidos por el resultado. La sintaxis es:

132 434 218 + CR..

652 132 OK

Primeramente introducimos una serie de números (un detalle: en FORTH los separadores son siempre los espacios) en la pila y ordenamos que se sumen los superiores. Por fin, nos imprime el resultado de la suma (que ha sustituido a los números sumados) y el otro número de la pila. Es como si con nuestros papeles cogiéramos los dos últimos los sumáramos y pusiéramos en lugar de ambos el resultado. De manera similar se hace con las demás operaciones: resta (–), multiplicación (*) y división (/).

Sin embargo, nuestro montón de papeles no es algo rígido e inmutable, sino que con cuidado podemos realizar algunos cambios, lo cual nos dará mayor facilidad de cálculo.

48 23 11 DUP * CR.

121 OK

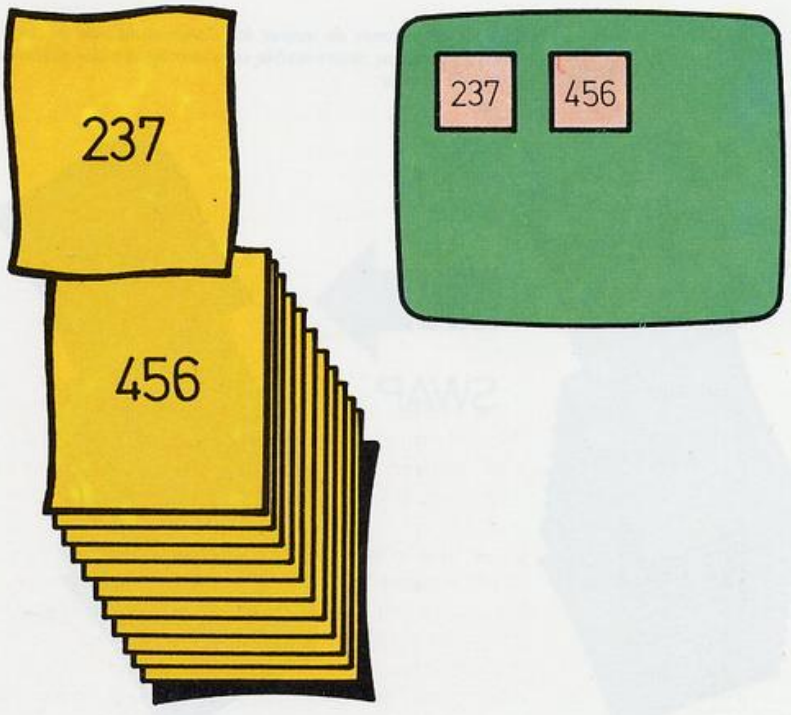
Aquí tenemos una palabra (recordemos que palabra es equivalente a comando u orden en FORTH) que nos permite realizar unos cambios en nuestra pila. En primer lugar introducimos tres números y al último de ellos le duplicamos (**DUP**), con lo cual ahora tendremos almacenados los números **48 23 11 11**. Por último se multiplican los dos números superiores y se sustituyen por el resultado, que es impreso obteniendo así el cuadrado de ese número. Con **DUP** es como si sacásemos una copia del papel que se encuentra arriba y lo depositásemos encima de todos.

También podemos coger el número superior y eliminarle sin hacer ninguna operación (lo contrario a **DUP**). Para ello utilizamos **DROP**. Con **OVER** sacamos una copia del penúltimo número de la pila y lo coloca encima de todos. Vamos a combinar las dos palabras para comprobar en la práctica su uso.

35 43 12 OVER DROP CR...

12 43 35 OK

Los datos se van almacenando unos encima de los otros y se extraen en orden inverso al de su introducción.

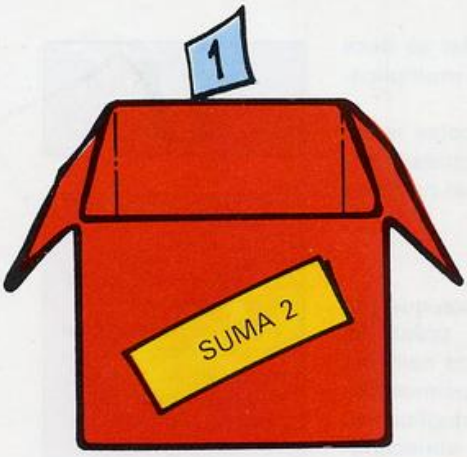


¡!

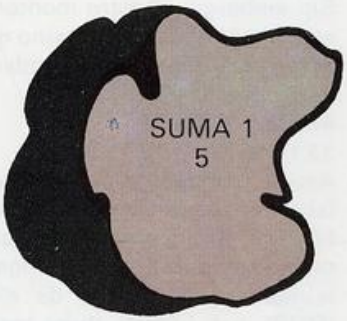
Cuando realizamos una operación (aritmética, visualización o de condición) con los datos de la pila, los que son utilizados se pierden y son sustituidos por el resultado, en el caso de la operación, o por un cero si la condición es falsa o un uno si es verdadera.

*

En la pila es donde el FORTH guarda los datos de donde los extraerá cuando sea necesario operar.



VARIABLE



CONSTANTE

bia las dos hojas de arriba y **ROT** que se atreve a cambiar de lugar los tres papeles superiores, de tal manera que el que se encuentra en tercer lugar (contando por arriba) le extrae y coloca encima de todos, por tanto el que estaba en segundo pasará a ser el tercero y el primero el segundo.

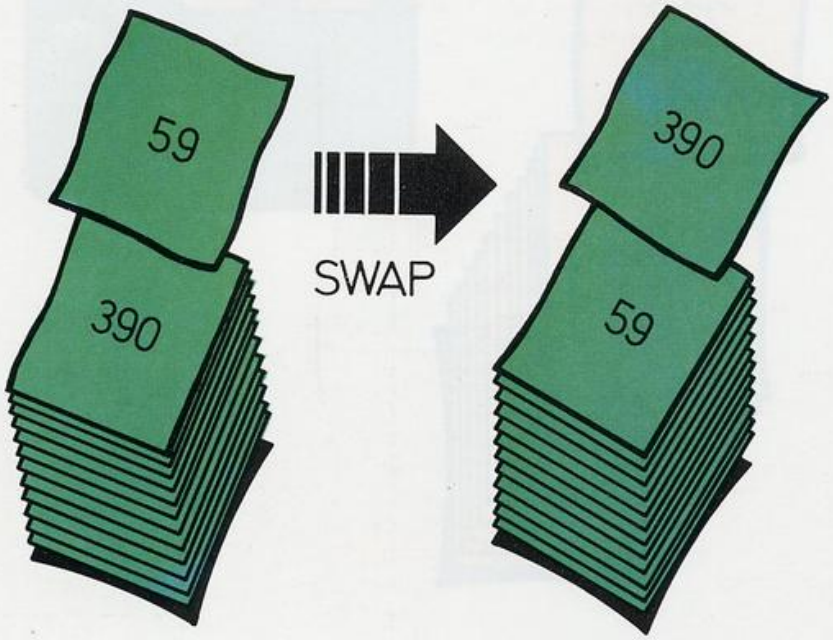
67 54 32 ROT CR...
54 32 67 OK
32 51 28 SWAP CR...
32 28 51 OK

CONSTANTES Y VARIABLES

El FORTH, al igual que en otros lenguajes, nos ofrece la posibilidad de guardar los datos en variables o en constantes.

Después de introducir los datos en la pila, **OVER** nos duplica el penúltimo dato (el 43) con lo que la pila quedará **35 43 12 43**. Sin embargo, **DROP** elimina el último de la pila, por lo tanto tendremos otra vez la situación inicial **35 43 12**, como se puede comprobar al imprimirlos.
Podemos incluso cambiar la colocación de los números de la pila, pero claro, de los de arriba, para evitar que se nos caiga nuestro montoncillo de papeles. Para ello tenemos **SWAP** que intercam-

Una de las formas de mover los datos de la pila es con SWAP, que nos intercambia el orden de los dos últimos datos de la pila.



FORTH también conoce otras formas de guardar valores para luego trabajara con ellos, y que son comunes a otros lenguajes informáticos; son las constantes y las variables, pero es necesario decir que para operar con ellas siempre deben estar en nuestra pila de trabajo (conocida mejor por pila o *stack* de usuario).

5 CONSTANT SUMA1
1 VARIABLE SUMA2
SUMA 2 §
SUMA 1 +

Veamos en que consiste este ejemplo: en primer lugar definimos una constante y le damos el valor **5** que se almacena en **SUMA1**. Posteriormente hemos hecho lo mismo con una variable y su valor, **1**, se guarda en **SUMA2**. Lo siguiente es guardar el valor de la variable en la pila del usuario, algo necesario si queremos operar con ella, para lo cual nos valemos de la palabra **§**. Por último, entramos el valor de la constante en la pila y realizamos la suma de ambos.
Ahora tenemos dos opciones: o visualizar el resultado, con lo cual ya sabemos que lo perdemos de la pila, o volverlo a guardar en la variable para cuando nos sea necesario usar de nuevo su valor. Como lo primero ya sabemos hacerlo vamos a efectuar la segunda opción. Teclee ahora:

SUMA2 !

Ya tenemos guardado el valor superior de la pila en la variable **SUMA2**. ¿Qué no nos lo creemos? Pues a teclear:

SUMA2 §.

O bien:

SUMA2 ?

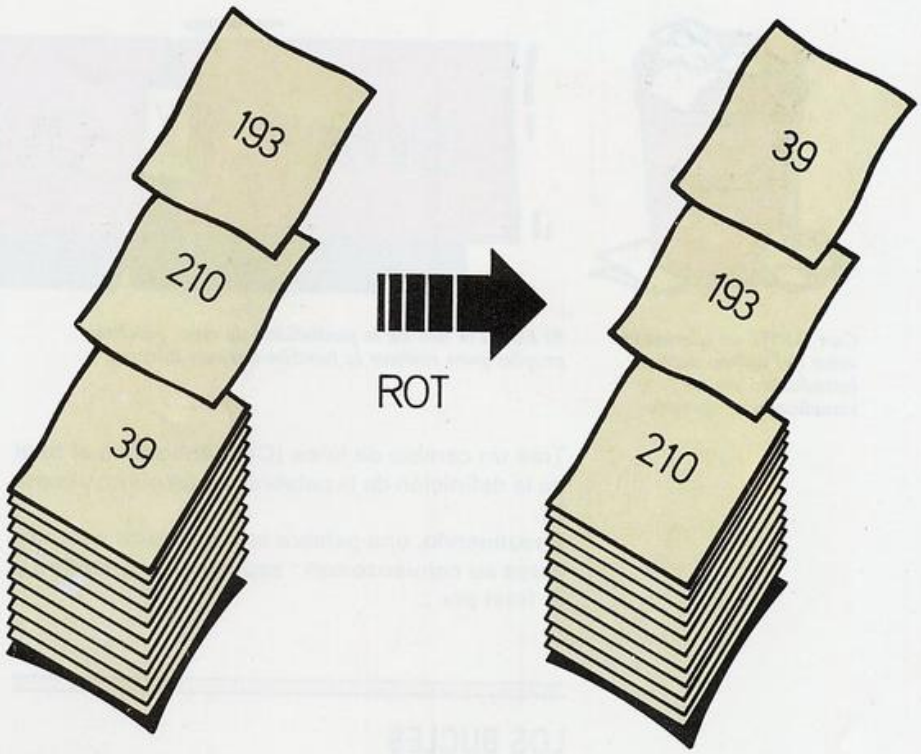
En ambos casos aparecerá «6», que es el valor esperado. Ambas expresiones son equivalentes, con lo cual la segunda se explica por la primera.

DEFINIMOS NUESTRAS PALABRAS

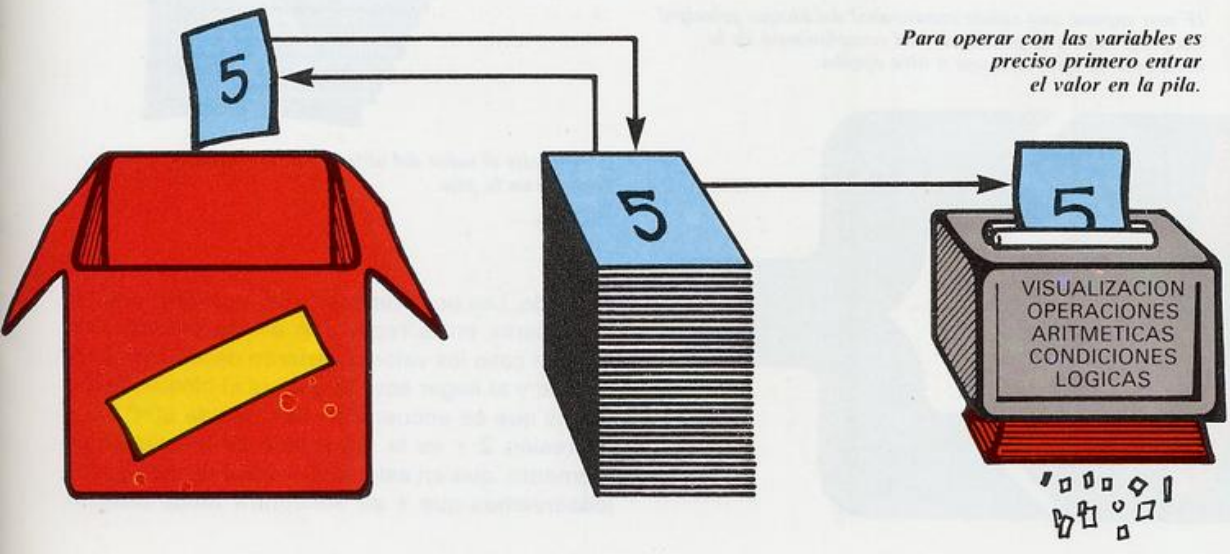
Ya comentamos al principio del artículo que el FORTH trabaja con palabras, de tal manera, que no se considera que existan programas en FORTH sino palabras. Además podemos crear nosotros nuestras propias palabras para realizar todas aquellas operaciones necesarias.
Podremos definir palabras nuevas que podrán estar formadas por otras (ya sean también creadas por nosotros o de las que FORTH incorpora). Una vez definidas, con llamarles por su nombre basta para que se ejecuten y aparezcan los resultados esperados.
Pasemos a la práctica y definamos nuestra primera palabra:

```
:LETRA
CR KEY
DUP EMIT CR
3 + EMIT CR ;
```

Si suena a chino no nos preocupemos que ahora intentaremos que todo quede claro. En primer lugar vemos : cuya función es indicar que vamos a definir una palabra nueva a la cual llamaremos por el nombre indicado posteriormente **(LETRA)**. La función de la palabra **KEY** es esperar la pulsación de una tecla y guardar su código ASCII en la pila del usuario, trabajando con él como si fuera un número. Así, si hemos pulsado la **A** en la pila se guardará el número **65**. Tras introducir la tecla, la palabra continúa ejecutándose. Posteriormente, duplicamos el valor del último número de la pila, que será el código ASCII de la tecla pulsada. A continuación imprimimos la letra cuyo código se encuentre en la parte superior de la pila. Esto lo hacemos con ayuda de **EMIT**. Así, continuando con nuestro ejemplo, se imprimirá **A** porque en la pila tenemos su código.
El siguiente paso está claro, puesto que sumamos **3** a la pila, con lo cual el valor de ella pasa a ser ahora de **68**. Después le indicamos que imprima la letra cuyo código se encuentra en la parte superior de la pila, que, en este ejemplo es **D**.



ROT *también nos ofrece la posibilidad de intercambiar los valores de la pila. En este caso se intercambian los tres últimos datos del stack.*



Para operar con las variables es preciso primero entrar el valor en la pila.

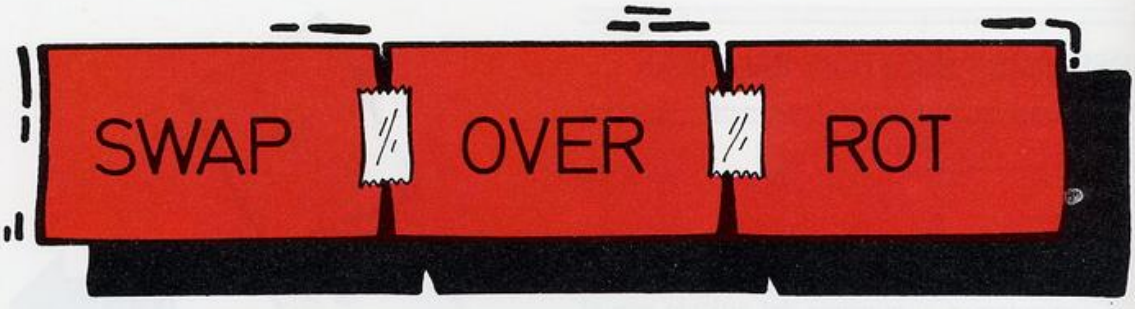
¡!

Los gráficos y el sonido son iguales que en el BASIC, salvo que los datos se anteponen a la palabra separados, por un espacio.

*

Para imprimir una cadena alfanumérica se escribe primero un punto (.), y, sin separación, la cadena entrecomillada cuyo primer carácter debe ser siempre un espacio.

Con **DROP** *se elimina el valor del último dato introducido sin ser visualizado ni operado.*



El FORTH nos da la posibilidad de crear palabras propias para realizar la función que nos interesa.

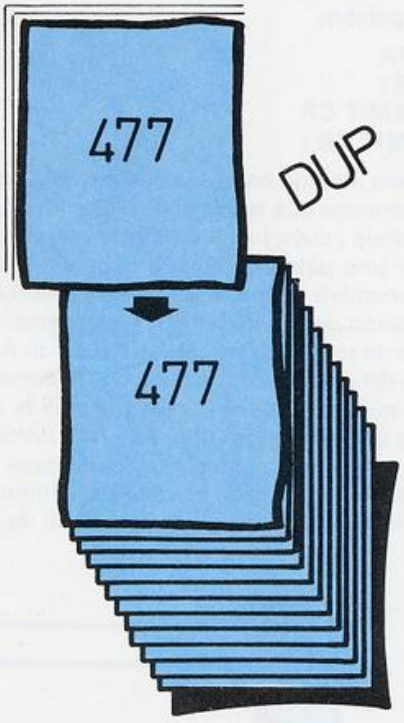
Tras un cambio de línea **(CR),** indicamos el final de la definición de la palabra con un punto y coma (;).
Resumiendo, una palabra se caracteriza por indicarse su comienzo con : seguido de su nombre y el final por ;.

LOS BUCLES

Dentro de este grupo general de órdenes, cuya característica común es provocar la repetición de un grupo de comandos, podemos distinguir los no condicionales y los condicionales (que a diferencia del grupo anterior, una condición limita la continuidad del bucle).
Dentro del primer grupo tenemos las palabras **DO...LOOP,** que se presentan siempre asociadas y entre ellas se introducen las instrucciones a repetir. Para una mejor comprensión diremos que son totalmente equivalentes a los bloques **"FOR/TO...NEXT"** del BASIC. Veamos el siguiente ejemplo:

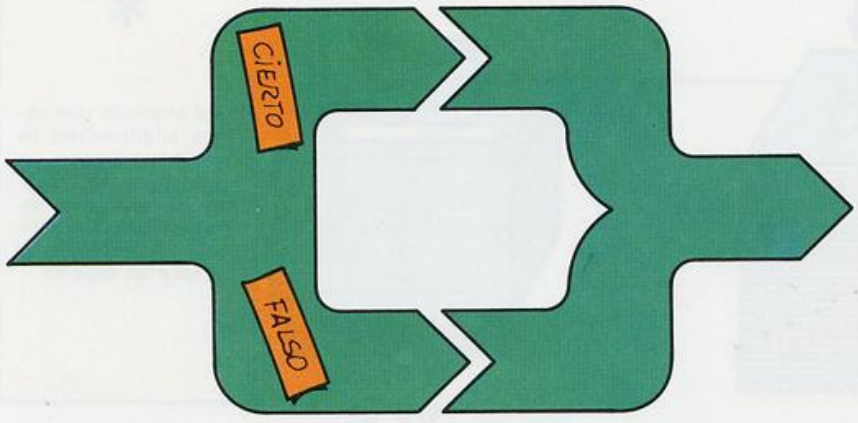
: PRESENTACION 6 0 DO ."SOY FORTH" CR 2 +LOOP ;

Al ejecutar esta palabra (llamada «PRESENTACION») podemos comprobar que se imprime tres veces el mensaje «SOY FORTH» debido al bucle utilizado. Los dos primeros números, **6 0,** son los limitadores en la repetición de las órdenes, así en este caso los valores variarán desde cero hasta seis, y al llegar aquí se saltará al bloque de palabras que se encuentran después de **LOOP.** La expresión **2 +** es la encargada de indicar el incremento, que en este caso lo hará de dos en dos (observemos que + se encuentra unido directa-



DUP *repite el valor del último dato introducido y lo deposita en la pila.*

IF nos supone una salida condicional del bloque principal del programa, y dependiendo del cumplimiento de la condición se ejecutará una u otra opción.



mente a **LOOP**). Se hubiera podido simplificar este ejemplo con haber hecho **3 0 DO** ... y haber suprimido el incremento.
Conviene fijarse en la forma que se debe emplear para imprimir cadenas alfanuméricas, sobre todo porque ninguna de las dos guías FORTH de los programas mencionados menciona como hacerlo. Después de . y **sin separación** se coloca la cadena entrecomillada y cuyo primer carácter debe ser siempre un espacio.
El FORTH dispone también de la palabra condicional **IF** que, salvo su sintaxis, es en todo equivalente al BASIC.

```
: IGUALDAD CR KEY CR KEY > IF
." ES MAYOR EL PRIMERO" ELSE
." ES MAYOR EL SEGUNDO" ENDIF ;
```

Los términos a comparar se deben colocar en la pila y justo antes de la palabra **IF** (aquí se comparan los dos datos introducidos por teclado) se indica el signo de la condición (<>=). Si se cumple, se ejecuta lo que está a continuación y en el caso contrario salta y es ejecutado lo que se encuentra después del **ELSE**. Por último, se indica el final del bloque **IF** con **ENDIF**.
Dentro de los bucles condicionales, propiamente dichos, tenemos dos opciones: **BEGIN ... UNTIL** y **BEGIN ... WHILE ... REPEAT**. La diferencia entre ambas se encuentra en que mientras en la primera el bucle se está ejecutando hasta que se cumpla la condición (es decir, que el número superior del *stack* sea un cero), en el segundo se ejecuta continuamente hasta que la condición deje de cumplirse. Veamos sus sintaxis:

```
: LETRA CR BEGIN KEY 65
— UNTIL ." NO HAS APRETADO LA A" ;
```

BEGIN indica el comienzo del bucle en donde se introduce una tecla y a cuyo código se le resta 65. Si se ha pulsado **A** el valor que queda en la parte superior de la pila es **0**, por tanto vuelve a ejecutarse el bucle. Cuando no se apriete la tecla salta y se imprime el mensaje del final.

La palabra KEY *espera la pulsación de una tecla e introduce su código ASCII en la parte superior de la pila.*

```
: OPERACIONES CR BEGIN 10 20
KEY 65 = WHILE DROP + ." 10+20=" .
REPEAT DROP * ." 10*20=" . ;
```

En este caso, dentro del bucle introducimos dos números, el código de una letra y un cuarto número. Cuando se cumpla la condición (pulsemos **A**) se imprimirá la suma de los dos números y se repetirá de nuevo el bucle. Si no se cumple se ejecuta lo que se encuentra después del **REPEAT** abandonando el bucle.

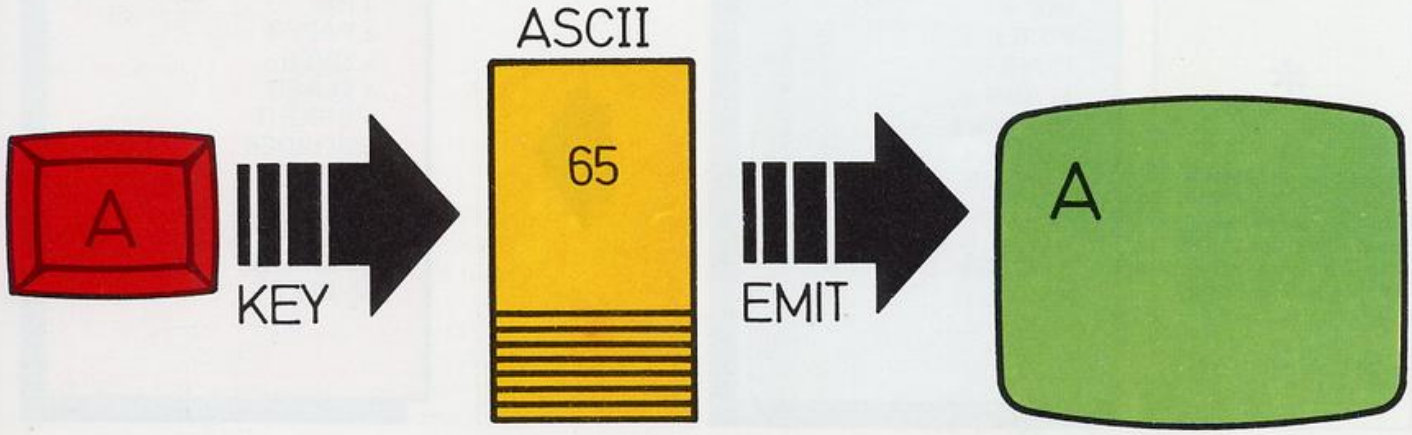
EMIT *es la contraria a* **KEY**: *imprime la letra cuyo código ASCII está indicado por el número que se encuentra en la parte superior de la pila.*

¡!

Una palabra de FORTH se indica porque en su comienzo se escribe : y posteriormente su nombre. Se termina la definición con ;.

*

Los bucles son de dos tipos: los condicionales y los no condicionales. Los primeros tienen la forma **DO ... LOOP** y los segundos pueden ser **BEGIN ... UNTIL** o **BEGIN ... WHILE ... REPEAT**.



¡!

Las peculiaridades o los inconvenientes más característicos de este lenguaje es que no «sabe» operar con decimales ni conoce otras operaciones que las cuatro básicas (+, –, *, /).

*

En **BEGIN** ... **UNTIL** el bucle se está ejecutando hasta que se cumpla la condición. En **BEGIN** ... **WHILE** ... **REPEAT** el bucle se ejecuta hasta que la condición deje de cumplirse.

*

Las funciones **KEY** o **EMIT** son contrarias; mientras que la primera espera a que sea pulsada una tecla y guarda el valor de su código ASCII en la pila, la segunda coge el dato superior del *stack*, lo toma como código ASCII e imprime el carácter correspondiente.

*

Tenemos la posibilidad de crear nuestras propias palabras formadas por otras creadas por nosotros o de la que el lenguaje incorpora.

La unica limitacion de los bucles no condicionales es el número de veces que se debe repetir las instrucciones.

LOS GRAFICOS Y LOS SONIDOS

Sería ilógico pensar que un lenguaje con la finalidad del FORTH no tuvira posibilidad gráfica y sonora. Pero si le ha parecido difícil lo visto hasta ahora, este es el momento de descansar dado que, en este aspecto, los comandos coinciden con los del BASIC SPECTRUM y la diferencia se encuentra que hay que anteponer los valores separados por un espacio. Lo único reseñable es que en la versión FORTH de *ABERSOFT* el comando **DRAW** no se refiere a los incrementos de X e Y (como es en el BASIC SPECTRUM) sino a la posición absoluta donde terminará el trazo.

Los comandos gráficos y sonoros son iguales a los del BASIC, sólo cambia la posición de los datos.

BASIC

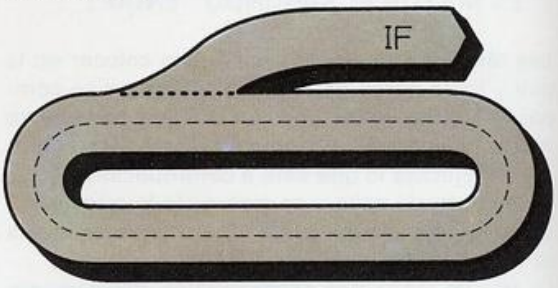
```
INK a
PAPER a
OVER a
FLASH a
BRIGHT a
BORDER a
INVERSE a
PLOT a, b
DRAW a, b
CIRCLE a, b
BEEP a,b
```

FORTH

```
a INK
a PAPER
a OVER
a FLASH
a BRIGHT
a BORDER
a INVERSE
ab PLOT
ab DRAW
ab CIRCLE
ab BEEP
```

YA TERMINAMOS

¿Qué le ha parecido esta introducción al FORTH-SPECTRUM?: cualquier cosa menos un camino de rosas. Este lenguaje a todas luces no es para los que se inicien en la informática, aunque sus autores propagaran que también servía para eso. Es indudable que introducir la idea de *stack* y de su manejo a un novicio se puede hacer bastante cuesta arriba.



En un bucle condicional hay una condición lógica que cuando se cumple se sale de él.

Con respecto a las dos versiones FORTH para Spectrum queda por decir que son bastante similares entre sí y, además de las ya comentadas, aportan de diferencias que la de INVESTRONICA tiene su guía en castellano y el Editor (parte del FORTH que sirve para facilitar la corrección y realización de las palabras) hay que cargarlo aparte del intérprete, mientras que en el de *ABERSOFT*, la guía es en inglés y el Editor se carga al mismo tiempo que el FORTH.

MOVIENDO 16 BITS

EN el capítulo anterior discutíamos en detalle el grupo de carga de 8 bits. Nuestro próximo objetivo consistirá en efectuar otro tanto, con las instrucciones encargadas de gestionar la transferencia de 16 bits entre registros dobles y la memoria de nuestro ordenador.
A ellas uniremos un grupo muy especial, el cual actúa sobre una importante zona de la RAM del Spectrum denominada *stack* o pila, encargada de conservar ciertos datos de interés durante la ejecución de un programa, para su posterior utilización.

PAR	dd	qq
BC	00	00
DE	01	01
HL	10	10
SP	11	—
AF	—	11

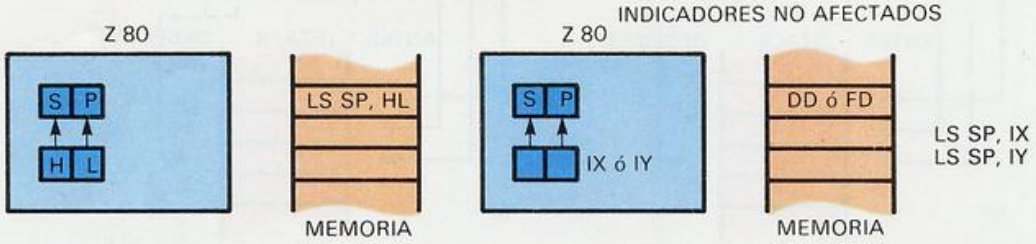
Decodificación seguida por el Z 80 para reconocer los pares de registros indicados.

CARGA DE 16 BITS

Si observamos la tabla que lleva este nombre al comienzo de nuestros capítulos dedicados al código máquina, la encontramos mucho menos poblada de lo que estaba la de carga de 8 bits, sobre todo en lo referente a transferencia de datos entre registros dobles. Solamente cuando el destinatario de la información es el *stack pointer* SP es posible seleccionar algunos pares de registros como origen de ésta.
La última fila y la última columna están señaladas con PUSH y POP, respectivamente, instrucciones que discutiremos en breve. Pero centrémonos en el resto de la tabla. Esta zona constituye el grupo de carga propiamente dicho y, por supuesto, el código mnemónico asociado es LD. El formato es idéntico al grupo anterior:

ENSAMBLADOR	CODIGO MAQUINA	N.º BYTES	CICLOS RELOJ	CICLOS MAQUINA
LD SP, HL	11111001	1	6	1
LS SP, IX	11011101 11111001	2	10	2
LS SP, IY	11111101 11111001	2	10	2

INDICADORES NO AFECTADOS



¡!

En la tabla de carga de 16 BITS la última fila y última columna están señaladas con PUSH y POP.

*

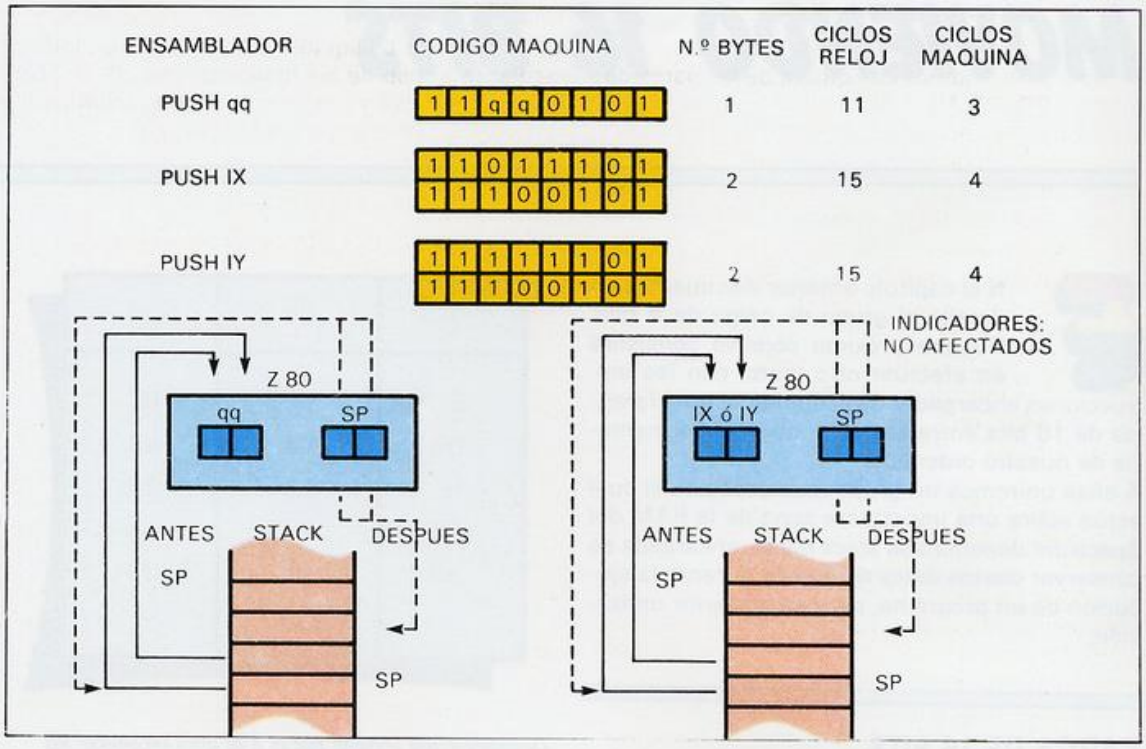
Las instrucciones PUSH y POP están ligadas con una zona de la memoria RAM de nuestro Spectrum denominada *stack* o pila.

¡!

Al grupo de registros dd es posible añadir los índices IX e IY, pues es factible almacenar directamente en ellos un entero nn.

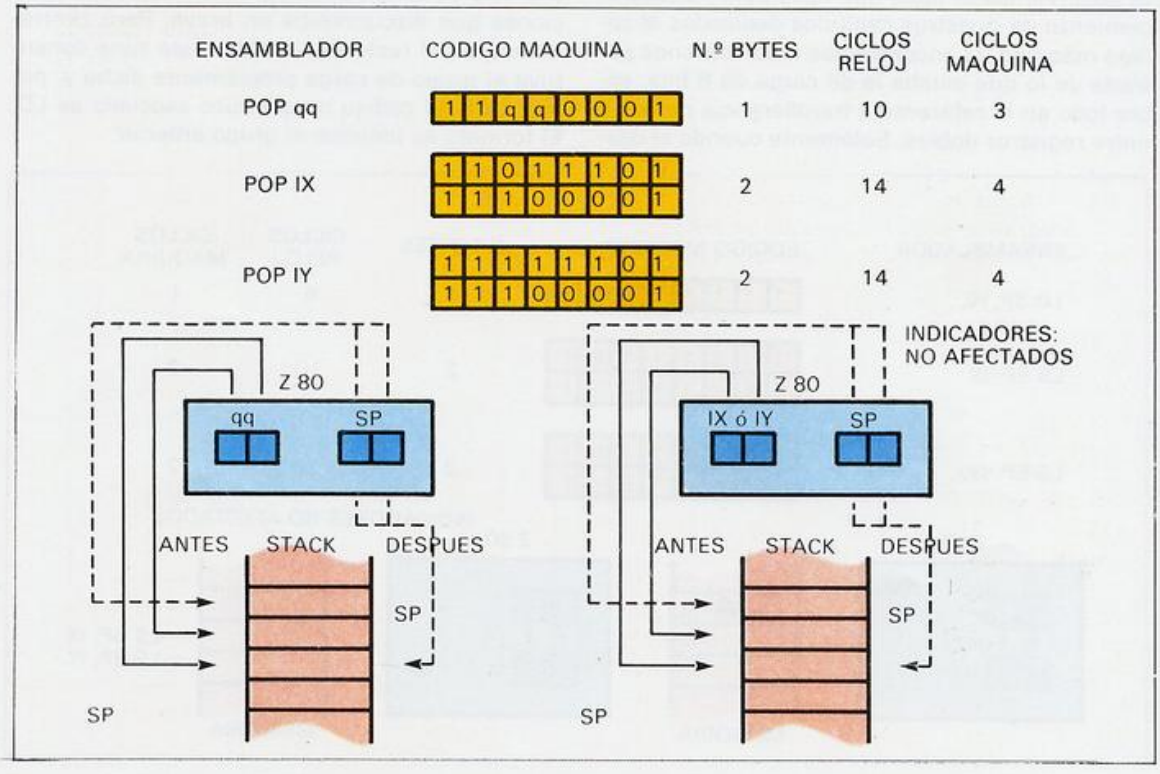
*

Con dd representamos cualquiera de los pares de registros BC, DE, HL o SP en la carga de registros inmediata extendida.



LD DESTINO,ORIGEN o LD DESTINO,DATO

pero con la diferencia, ya advertida, de que tanto ORIGEN, DESTINO, así como DATO son registros, posiciones de memoria o cantidades de 16 bits. Lo que es lo mismo: abarcan dos octetos.
Para encontrar el código hexadecimal correspondiente a una instrucción entraremos en la tabla por la fila DESTINO y buscaremos su intersección con la columna ORIGEN. De nuevo, si en este lugar se haya un cuadro en blanco, la instrucción planteada no está contemplada en el juego del Z80. Efectuemos la siguiente clasificación:



1. Carga en registros.
1.1. Inmediata extendida: **LD dd,nn**
Con dd representamos cualquiera de los pares de registros BC, DE, HL o SP. En la figura se muestra la decodificación seguida por la CPU de nuestro Spectrum para reconocerlos. Tras ejecutarla, el valor entero determinado por los dos bytes nn, es decir una cantidad comprendida entre 0 y 65535, se «copia» en el par de registros dd. Has de tener presente el orden de los octetos nn: primero el de menor peso y en segundo lugar el más significativo.
A los registros dd es posible añadir los índice IX e IY, pues es factible almacenar directamente en ellos un entero nn mediante las instrucciones **LD IX,nn** y **LD IY,nn.**
Como en el capítulo anterior, en las tablas correspondientes está recogida toda la información accesoria sobre cada grupo particular de instrucciones.
1.2. Carga extendida LD dd,(nn)
Para analizar el comportamiento de esta instrucción, lo mejor es recurrir a un ejemplo en el cual consideraremos que el par de registros dd es el BC. Supongamos, por tanto, que deseamos procesar **LD BC,(5FA6h)** o en su correspondiente expresión decimal **LD BC,(24486d).** Efectuemos también el supuesto de que la posición de memoria 24486 almacena el valor 176 y la 24487, conserva un 209.
Pues bien, tras leer el código de operación de dicha instrucción, el microprocesador procederá de la manera siguiente: buscará la posición de memoria 24486 y trasladará el dato allí almacenado (176), al octeto menos significativo del par de registros destino.
A continuación, volverá a acceder a la memoria para recoger el dato conservado en la dirección siguiente, es decir, en la 24487 y lo copiará en el byte de mayor peso del par BC, el cual, tomará el valor 176+209*256=53680.
Puedes comprobarlo tecleando y ejecutando el siguiente programa, el cual introduce en las direcciones del ejemplo los valores señalados y a continuación almacena en la memoria la rutina en C/M:

Dirección	Ensamblador
24488	LD BC,(24486d)
24492	RET

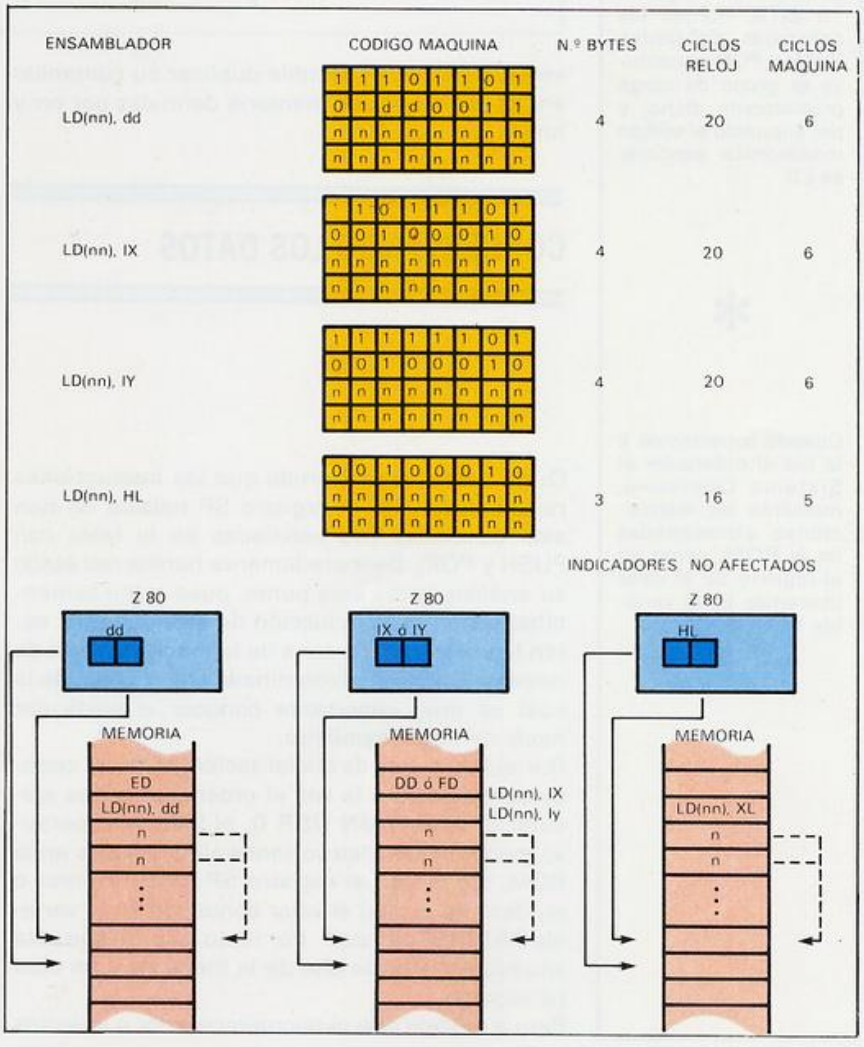
```
10 FOR I=24486 TO 24492
20 READ A: POKE I,A:NEXT I
30 PAUSE 0: PRINT USR 24488
40 DATA 176,209,237,75,166,95,201
```

Las instrucciones de este tipo ocupan cuatro bytes, pero dentro del juego del Z80, sus fabricantes decidieron dar un trato preferencial a la pareja HL, de modo tal, que cuando dd coincide con el par anterior (LD HL,(nn)) puede implementarse en sólo tres bytes. Por tanto, es ejecutable en dos formatos distintos, pero en este último se ahorra memoria y tiempo.

También es factible que el destinatario de la información sea cualquiera de los registros índice mediante el uso de las instrucciones **LD IX,(nn)** o **LD IY,(nn)** cuyo funcionamiento es idéntico al de las consideradas en este subgrupo.

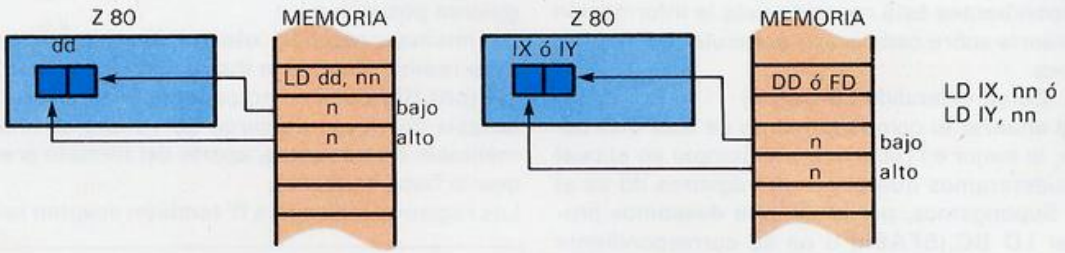
2. Carga en memoria.
2.1. Carga extendida: **LD (nn),dd**

Este grupo puede considerarse inverso del 2.2, el cual manejaba direccionamiento extendido para copiar la información almacenada en una pareja de registros en la memoria de nuestro ordenador. Ahora el proceso es el contrario, es decir, la CPU toma el byte de menor peso contenido en el par dd y lo duplica en la celda de memoria direccionada por nn. A continuación, lee el octeto más significativo conservado en dd y lo copia en la siguiente posición a nn.
La misma precisión efectuada anteriormente cabe realizar cuando la instrucción a procesar es **LD (nn),HL,** pues como podemos comprobar en la tabla del grupo de carga de 16 bits es implementable en tres bytes, aparte del formato previo que lo hace en cuatro.
Los registros índice IX e IY también aceptan la in-



ENSAMBLADOR	CODIGO MAQUINA	N.º BYTES	CICLOS RELOJ	CICLOS MAQUINA
LD dd, nn	0 0 0 0 0 0 0 1 / n n n n n n n n / n n n n n n n n	3	10	3
LD IX, nn	1 1 0 1 1 1 0 1 / 0 0 1 0 0 0 0 1 / n n n n n n n n / n n n n n n n n	4	14	4
LD IY, nn	1 1 1 1 1 1 0 1 / 0 0 1 0 0 0 0 1 / n n n n n n n n / n n n n n n n n	4	14	4

INDICADORES: NO AFECTADOS



¡!

La tabla de carga de 16 BITS, menos las columnas señaladas PUSH y POP, constituye el grupo de carga propiamente dicho, y por supuesto el código mnemónico asociado es LD.

*

Cuando conectamos a la red el ordenador el Sistema Operativo, mediante las instrucciones almacenadas en la **ROM**, carga en el registro SP el valor contenido en la variable RAMTOP-1.

versa, es decir, es factible duplicar su contenido en las posiciones de memoria definidas por nn y nn+1.

CONSERVANDO LOS DATOS

Quizá habremos advertido que las instrucciones relacionadas con el registro SP todavía no han sido discutidas (las señaladas en la tabla con PUSH y POP). Deliberadamente hemos retrasado su análisis hasta este punto, pues como comentábamos en la introducción de este capítulo, están ligadas con una zona de la memoria RAM de nuestro Spectrum denominada *stack* o pila, de la cual es muy importante conocer el particular modo de funcionamiento.

Durante la rutina de inicialización, es decir, cuando conectamos a la red el ordenador o tras ejecutar la orden **RUN USR 0**, el Sistema Operativo mediante las instrucciones almacenadas en la ROM, cargan en el registro SP *(Stack Pointer* o puntero de la pila) el valor contenido en la variable RAMTOP menos 1. Por tanto, al principio está situado en la parte alta de la memoria y no ocupa espacio.

Pero a medida que el microprocesador o nosotros mismos vamos introduciendo datos en él, «crece» desde allí hacia las direcciones de memoria menores, es decir, el valor del registro doble SP disminuye a medida que el *stack* aumenta de tamaño, y viceversa.

Esta circunstancia debe ser tenida muy en cuenta en la construcción de programas, pues un aumento incontrolado del *stack* puede traer como consecuencia la invasión por su parte del área de BASIC u otras zonas que en principio no hemos previsto para él, quedando adulterados por este motivo los resultados previstos.

Sobre la pila es posible efectuar dos operaciones: INTRODUCIR bytes (PUSH) o EXTRAERLOS (POP). Para ejecutarlas es necesario especificar siempre un par de registros como origen o destino de los datos intercambiados. Por tanto (es imprescindible tenerlo en cuenta), la transferencia de información entre registros y la pila o en sentido contrario es siempre de DOS en DOS bytes.

En la figura podemos encontrar los formatos bajo los cuales es posible implementar las instrucciones PUSH y POP. Veamos las operaciones llevadas a cabo por el Z80 al ejecutarlas. Supongamos que deseamos introducir (PUSH) un par de datos en la pila:

1) Se disminuye en 1 el contenido del par SP.
2) Se copia el contenido del byte de orden alto del par de registros origen en la posición de memoria indicada por SP.
3) Se decrementa SP.
4) Finalmente, el valor almacenado en el octeto de menor peso del par origen se duplica en la dirección apuntada por SP.

Ahora supongamos que queremos realizar el proceso inverso, es decir, extraer (POP) información de la pila para ser almacenada en un par de registros:

1) El contenido de la posición de memoria direccionada por SP se carga en el octeto de orden bajo del par de registros elegido.
2) Se le añade 1 al contenido del SP.
3) El byte indicado ahora por SP se transfiere al octeto más significativo del par correspondiente.
4) Nuevamente es incrementado en 1 el contenido del registro SP.

En la jerga informática esta forma de tratar los datos se denomina LIFO *(Last In, First Out:* último en entrar, primero en salir) y su utilidad es obvia si consideramos una cierta cantidad de información, la cual desborde la capacidad de los registros, y además, la deseemos mantener estructurada de manera tal que tengamos fácil y rápido acceso al recuperarla.

Algunos ordenadores disponen de la denominada pila del usuario para todas estas labores. Desgraciadamente el Spectrum sólo cuenta con la pila de máquina, y sobre ella actuamos nosotros con las instrucciones PUSH y POP.

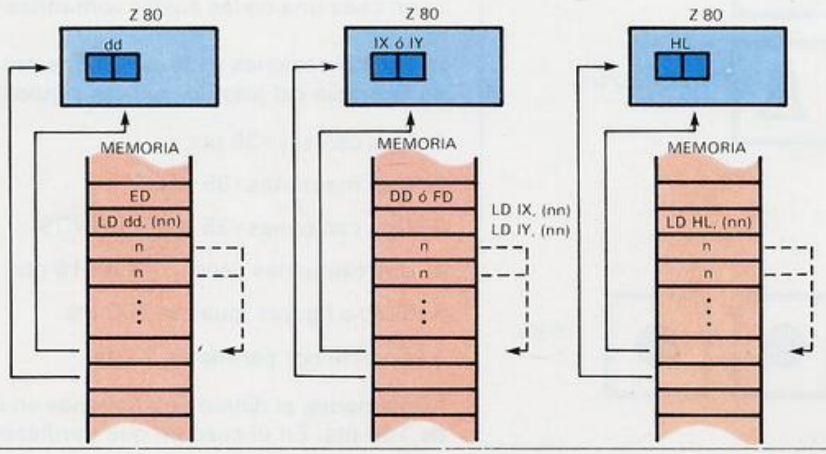
Pero el microprocesador también necesita almacenar ciertos valores de interés para él, por ejemplo, la dirección a donde regresar cuando termine de ejecutar una subrutina. Por ello, cuando hagamos uso de la pila debemos asegurarnos que el número de instrucciones PUSH y POP ejecutadas es el mismo.

En otro caso el Z80 tomará los dos bytes indicados por el registro SP como dirección de retorno y el resultado puede convertirse en imprevisible. Por otra parte, de la misma manera que en BASIC al ejecutar el comando **CLEAR** se modifica el RAMTOP del Sistema, en C/M es posible desplazar el puntero del stack SP a la posición de memoria determinada por un par de registros. Para ello basta ejecutar cualquiera de las instrucciones siguientes: **LD SP,HL, LD SP,IX** ó **LD SP,IY.**

La primera, como podemos comprobar en la tabla correspondiente es una instrucción rápida que tan solo ocupa un byte de memoria. Tras ejecutarla, el SP señalará a la posición de memoria definida por el contenido del par HL. Con esto queda cerrada la discusión emprendida en el capítulo anterior sobre las instrucciones de carga.

ENSAMBLADOR	CODIGO MAQUINA	N.º BYTES	CICLOS RELOJ	CICLOS MAQUINA
LD dd, (nn)	1 1 1 0 1 1 0 1 / 0 1 d d 1 0 1 1 / n n n n n n n n / n n n n n n n n	4	20	6
LD IX, (nn)	1 1 0 1 1 1 0 1 / 0 0 1 0 1 0 1 0 / n n n n n n n n / n n n n n n n n	4	20	6
LD IY, (nn)	1 1 1 1 1 1 0 1 / 0 0 1 0 1 0 1 0 / n n n n n n n n / n n n n n n n n	4	20	6
LD HL, (nn)	0 0 1 0 1 0 1 0 / n n n n n n n n / n n n n n n n n	3	15	5

INDICADORES: NO AFECTADOS



¡!

En la carga en extendida en memoria, la CPU toma el byte de menor peso, contenido en el par dd y lo duplica en la celda de memoria direccionada por nn.

*

Los registros índice IX e IY aceptan la inversa, es decir, es factible duplicar su contenido en las posiciones de memoria definidas por nn y nn+1.

TRAGAPERRAS

ESTE programa es la solución al problema que padecemos la casi totalidad de la humanidad: no nos gusta perder dinero en el juego. Con TRAGAPERRAS, podremos arriesgar nuestros ahorros, sin perder un solo duro utilizando el dinero imaginario que el Spectrum pone a nuestra disposición.

Para que cada una de las figuras comience a girar es necesario pulsar **ENTER** *cuando lo indique el ordenador.*

EL PROGRAMA

TRAGAPERRAS es un sencillo programa en el cual predomina la suerte y la aleatoriedad. Su manejo es muy sencillo; basta con seguir al pie de la letra las instrucciones que nos surjan en cada momento en la pantalla.

¿El objetivo del juego? No puede ser más simple; simula justamente lo que su propio nombre indica: una máquina tragaperras de casino.

El programa nos muestra cuatro ventanas en las cuales irán apareciendo en secuencia aleatoria, y en un espacio de tiempo determinado, una serie de figuras; dependiendo del orden final de las cuales obtendremos más o menos ingresos o... nada si la suerte no nos acompaña.

Para que cada una de las figuras comience a girar es necesario pulsar **ENTER** cuando así lo indique el ordenador. Una vez pulsada dicha tecla, el programa principal pone en funcionamiento una rutina que durante un intervalo de tiempo predeterminado irá seleccionando aleatoriamente en cada una de las cuatro ventanitas una figura.

Las combinaciones a las que el Spectrum concede la gracia del premio, son las siguientes:

1: Tres cerezas=35 pts.

2: Tres manzanas=35 pts.

3: Tres campanas=35 pts. + BONOS

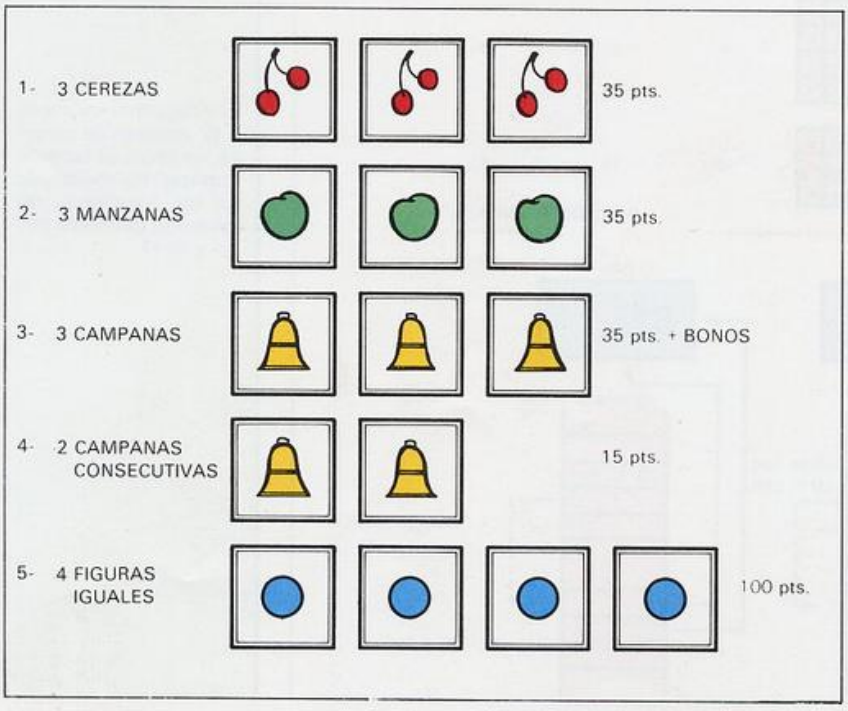
4: Dos campanas consecutivas=15 pts.

5: Cuatro figuras iguales=100 pts.

* El precio por partida es 15 pts.

Inicialmente, el dinero que tenemos en cuenta es de 100 pts. En el caso en que perdiésemos todo

Las combinaciones a las cuales el Spectrum concede la gracia de premio son las representadas en el dibujo.



el dinero acumulado, podremos volver a probar suerte con solo pulsar una tecla. Si por el contrario la suerte nos favorece, podremos seguir jugando siempre y cuando no sobrecalentemos los circuitos de la máquina tragaperras. Esto nos puede sonar a chiste, pero a ver quien es el guapo que se atreve a pasar el tope de 9.500 pts. que el Spectrum se ha autoimpuesto.

```
 10 REM ************************
 20 REM * J.M.MAYORAL SERRANO *
 30 REM ************************
 40 REM * TRAGAPERRAS    1985 *
 50 REM ************************
 60 REM [INV.] INIC VARIABLES [TRUE]
 70 RANDOMIZE
 80 LET SW=0
 90 PAPER 0: BORDER 0: INK 9
100 CLS
110 LET DIN=100
120 DIM A$(3,4): DIM B$(3,4): DIM A(4): DIM Q(4)
130 IF SW THEN  GO TO 200
140 PRINT AT 10,5;"UN MOMENTO POR FAVOR"
150 GO SUB 1250
160 CLS
170 FOR N=1 TO 4
180 PRINT  PAPER 2; INK 6;AT 4,N*3;N
190 NEXT N
200 REM [INV.] DIBUJO MAQUINA [TRUE]
210 FOR M=22 TO 98 STEP 24
220 PLOT M,129: DRAW 19,0
```

```
 230 DRAW 0,-19
 240 DRAW -19,0
 250 DRAW 0,19
 260 NEXT M
 270 PLOT 18,131
 280 DRAW 99,0
 290 DRAW 0,-23
 300 DRAW -99,0
 310 DRAW 0,23
 320 PLOT 14,98
 330 DRAW 107,0
 340 DRAW 0,-13
 350 DRAW -107,0
 360 DRAW 0,13
 370 PLOT 10,50
 380 DRAW 0,100
 390 DRAW 10,6
 400 DRAW 95,0
 410 DRAW 10,-6
 420 DRAW 0,-100
 430 DRAW -115,0
 440 PRINT AT 12,4;"FRUIT"
 450 PRINT AT 14,6;"MACHINE"
 460 PRINT AT 10,2; INK 6;"8888888888888"
 470 GO SUB 720
 480 IF RND>.87 AND PR<100 THEN  GO SUB 500
 490 GO TO 470
 500 REM [INV.] AVANCES [TRUE]
 510 FOR T=1 TO 7: BEEP .1,50-T*7: NEXT T
 520 PRINT AT 8,19; FLASH 1; INK 9;"AVANCES"
 530 LET CONTADOR=0
 540 PRINT AT 21,2;"PULSA EL NUMERO DE CASILLA"
 550 IF CONTADOR=4 THEN  PRINT AT 21,0;"
                    ": GO TO 640
 560 INPUT Q$
 570 IF Q$="" THEN  LET CONTADOR=4: GO TO 550
 580 IF CODE Q$<49 OR CODE Q$>52 THEN  BEEP 1,-10: GO
 TO 560
 590 LET CONTADOR=CONTADOR+1
 600 LET Q=VAL Q$(1)
 610 LET Q(Q)=Q
 620 PRINT AT 9+Q,18; PAPER RND*4; INK 9; FLASH 1;"PA
RADO >";Q;" "
 630 GO TO 560
 640 PRINT AT 8,18;"                ";AT 10,18;"
       ";AT 11,18;"                ";AT 12,18;"
      ";AT 13,18;"
 650 RETURN
 660 REM [INV.] COG. NUM. [TRUE]
 670 PRINT AT 6,3;A$(A(1));AT 7,3;B$(A(1))
 680 PRINT AT 6,6;A$(A(2));AT 7,6;B$(A(2))
 690 PRINT AT 6,9;A$(A(3));AT 7,9;B$(A(3))
 700 PRINT AT 6,12;A$(A(4));AT 7,12;B$(A(4))
 710 RETURN
 720 REM [INV.] NUEVO JUG. [TRUE]
 730 LET DIN=DIN-15
 740 LET PR=0
 750 LET O$="": LET P$=""
 760 PRINT AT 4,18; FLASH 1; PAPER 1; INK 7;"PULSA EN
TER"
 770 IF INKEY$<>"" THEN  GO TO 770
 780 IF INKEY$<>CHR$ 13 THEN  GO TO 780
 790 PRINT AT 4,18;"                ";AT 5,19;"
    ";AT 6,21;"
 800 PRINT AT 19,0;"                                "
 810 PRINT AT 20,0;"                                "
 820 PRINT AT 10,2; PAPER 6;"                "
 830 PRINT AT 2,18; PAPER 1; INK 9;"DINERO =  ";DIN;
PAPER 0;"  "
 840 FOR Z=1 TO 8
 850 FOR B=1 TO 4
 860 IF Q(B)=B THEN  GO TO 890
 870 BEEP .005,40
 880 LET A(B)=INT (RND*3)+1
 890 NEXT B
 900 GO SUB 660
 910 NEXT Z
 920 DIM Q(4)
 930 IF A(1)=A(2) AND A(2)=A(3) AND A(3)=A(4) THEN  L
ET Q$=" EL ESPECIAL!! ": LET PR=100: GO TO 960
 940 IF (A(1)=A(3) AND A(3)=A(4)) OR (A(1)=A(2) AND A
(2)=A(3)) OR (A(2)=A(3) AND A(3)=A(4)) OR (A(1)=A(2)
AND A(2)=A(4)) THEN  LET Q$=" 3 IGUALES!! ": LET PR=3
5
 950 IF A(1)=2 AND A(2)=2 OR A(2)=2 AND A(3)=2 OR A(3
)=2 AND A(4)=2 THEN  LET P$=" BONOS!! ": LET PR=PR+15
 960 FOR G=1 TO 15
 970 BEEP .008,50-G: BEEP .008,50
 980 NEXT G
 990 LET DIN=DIN+PR
1000 IF Q$="" THEN  GO TO 1070
1010 PRINT AT 19,6; BRIGHT 1; FLASH 1; INK 2; PAPER 4
;Q$
1020 PRINT TAB 6; BRIGHT 1; FLASH 1; INVERSE 1; INK 2
; PAPER 4;Q$
1030 IF PR>0 THEN  PRINT AT 10,4; BRIGHT 1; INK 2;"GA
NAS $";PR
1040 IF P$="" THEN  GO TO 1070
1050 PRINT AT 10,20; FLASH 1; INK 5; PAPER 3;P$
1060 PRINT AT 11,20; INVERSE 1; FLASH 1; INK 3;P$
1070 IF PR=0 THEN  GO TO 1170
1080 LET PP=0
1090 LET BOR=INT (RND*6)+2: FOR N=1 TO 50
1100 BEEP .01,40: BORDER BOR: BORDER 0
1110 NEXT N
1120 BORDER 0
1130 PRINT AT 2,28; FLASH 1; INVERSE 1; PAPER RND*4;
INK 9;DIN
1140 PRINT AT 19,6;"                ";TAB 6;"
"
1150 PRINT AT 10,2; INK 6;"8888888888888"
1160 PRINT AT 10,20;"            ";AT 11,20;"
"
1170 IF DIN<15 AND PR=0 THEN  PRINT AT 21,1; FLASH 1;
" NO TIENES DINERO PARA JUGAR ": PRINT #0;"  PULSA UN
A TECLA PARA EMPEZAR": GO TO 1190
1180 GO TO 1210
1190 IF INKEY$="" THEN  GO TO 1190
1200 RUN
1210 IF DIN>9500 THEN  PRINT AT 20,0; PAPER 2; INK 9;
 FLASH 1;"      ACABAS  DE  FUNDIRME!!     ": PRINT #0
;" PULSA UNA TECLA PARA COMENZAR ": GO TO 1230
1220 RETURN
1230 IF INKEY$="" THEN  GO TO 1230
1240 RUN
1250 REM [INV.] GRAFICOS USUARIO [TRUE]
1260 FOR Z=0 TO 7
1270 READ A,B,C,D
1280 POKE USR "A"+Z,A
1290 POKE USR "B"+Z,B
1300 POKE USR "C"+Z,C
1310 POKE USR "D"+Z,D
1320 NEXT Z
1330 LET A$(1)=CHR$ 16+CHR$ 2+"A"+"B"
1340 LET B$(1)=CHR$ 16+CHR$ 2+"C"+"D"
1350 FOR Z=0 TO 7
1360 READ E,F,G,H
1370 POKE USR "E"+Z,E
1380 POKE USR "F"+Z,F
1390 POKE USR "G"+Z,G
1400 POKE USR "H"+Z,H
1410 NEXT Z
1420 LET A$(2)=CHR$ 16+CHR$ 4+"E"+"F"
1430 LET B$(2)=CHR$ 16+CHR$ 4+"G"+"H"
1440 FOR Z=0 TO 7
1450 READ J,K,L,M
1460 POKE USR "J"+Z,J
1470 POKE USR "K"+Z,K
1480 POKE USR "L"+Z,L
1490 POKE USR "M"+Z,M
1500 NEXT Z
1510 LET A$(3)=CHR$ 16+CHR$ 3+"J"+"K"
1520 LET B$(3)=CHR$ 16+CHR$ 3+"L"+"M"
1530 RETURN
1540 DATA 0,8,121,254,0,28,63,254,0,48,63,254,15,32,3
1,254,31,240,15,252,51,252,7,248,52,254,1,224,101,254
,0,0,0,128,7,240,0,128,15,248,1,192,15,248,3,224,31,2
52,3,224,0,128,3,224,1,192,3,224,1,192,3,224,0,0,0,0,
1,136,0,2,26,8,56,12,60,16,124,52,126,208,127,204,61,
224,124,20,25,240,124,36,1,224,56,72,0,224
```