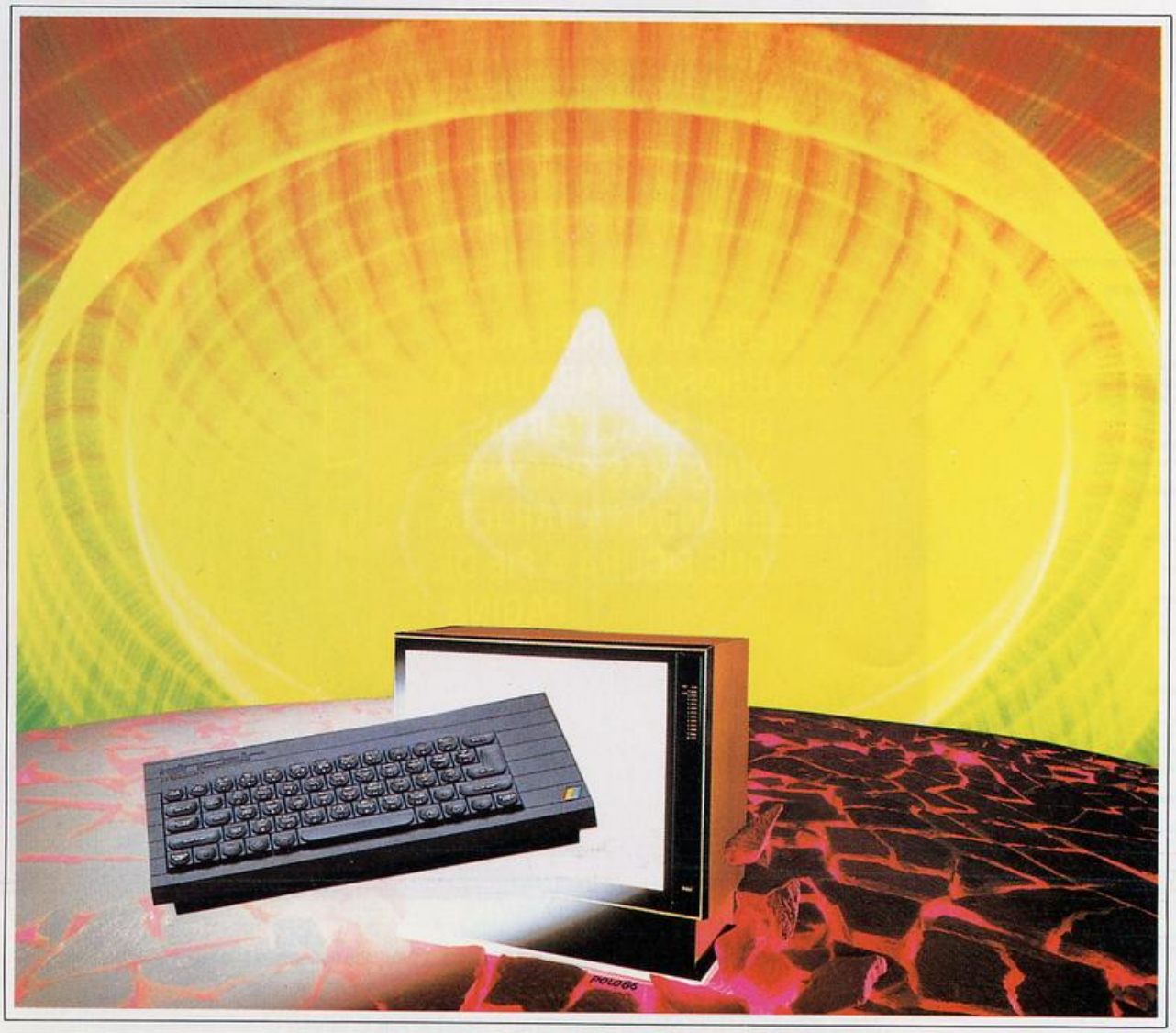
48
185 pts.
IVA INCLUIDO

RUN

Enciclopedia Práctica del Spectrum



Nueva Lente/Ingelek

ENSAMBLADORES

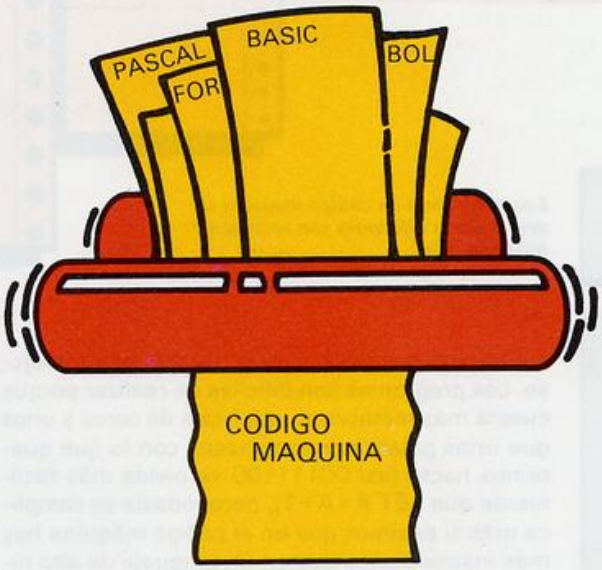
Si echamos un vistazo al mundo informático actual, nos daremos cuenta de la gran cantidad de lenguajes que nos rodean: empezando por diferentes dialectos del BASIC, uno por cada máquina (o incluso algunos ordenadores tienen posibilidad de incorporar diferentes versiones de BASIC por *software,* como es el caso del SPECTRUM), y terminando por las diferentes versiones de otros lenguajes con nombres de claves secretas de espías.

Si profundizamos un poco más en la inmensa «Torre de Babel» informática nos percataremos que cada lenguaje ha sido creado para un uso concreto: BASIC, para aprender; COBOL, para el manejo de gran cantidad de información; LOGO, para los niños; PASCAL, el superestructurado; FORTRAN, para los cálculos matemáticos... Sin embargo, por muchos lenguajes que nos rodeen y nos amenacen a todos, en el fondo, y para la máquina, se reducen a uno: el lenguaje máquina. Cuando estamos trabajando con un lenguaje de alto nivel, en el momento de ejecutarlo es necesario que el «micro» lo traduzca a código máquina, que es lo único que entiende nuestra Z-80. El proceso de traducción, sobre todo si el lenguaje que estamos utilizando es un intérprete, es el que retrasa todo programa no siendo lo suficientemente rápido para ciertas aplicaciones. Entonces ¿por qué no nos animamos a programar en código máquina?

Si ya nos hemos recuperado del susto por la anterior frase desafiante, intentaremos ver de una forma objetiva si es conveniente o no realizar un programa en código máquina.

Entre las ventajas que nos aporta, la más llamativa es su velocidad. Un programa en este lenguaje es evidente que no necesita traducción, porque ya le hablamos a la máquina en el lenguaje que le gusta, y por tanto tarda bastante menos en ejecutar un programa (en algunos casos la velocidad es 60 veces superior a la del BASIC). Otra de las ventajas del lenguaje máquina es la poca memoria que ocupan los programas, y esto se nota sobre todo en los más largos.

Pero no todo iba a ser un camino de rosas, también hay inconvenientes. En caso de error de programación nos costará hallarlo, puesto que no tenemos mensajes de error, que siempre nos pueden ayudar a solucionar los problemas. Otra característica a apuntar es que aunque cada ordenador utilice BASIC's diferentes, con muy poco conocimiento de las particularidades de la máquina podemos adaptar programas para que funcionen en diferentes «micros»; sin embargo esto no es posible con el lenguaje máquina, porque nos obliga a conocer a fondo el funcionamiento



Para el ordenador todos los lenguajes son iguales, ya que se reducen al código máquina.

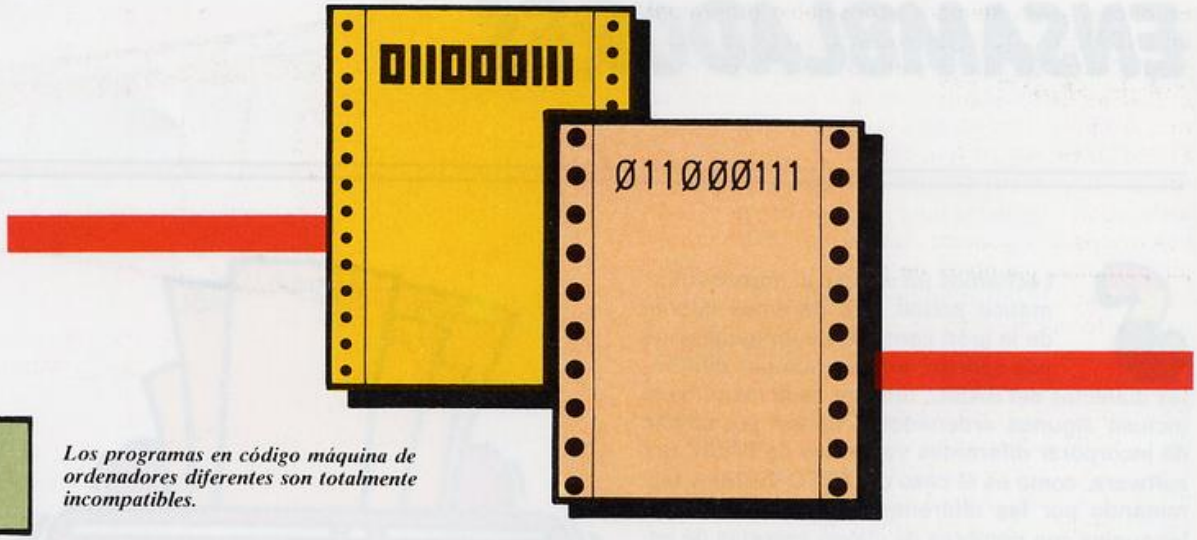
En la ejecución de un programa se pierde mucho tiempo en la traducción.

¡!

Cada lenguaje ha sido creado para un uso concreto, pero todos, en el fondo, se reducen a uno: el lenguaje máquina.

*

El proceso de traducción de un lenguaje de alto nivel es el que retrasa la ejecución de un programa, no siendo lo suficientemente rápido para ciertas aplicaciones.



Los programas en código máquina de ordenadores diferentes son totalmente incompatibles.

¡!

Los inconvenientes es la ausencia de los mensajes de error, la falta de compatibilidad entre diferentes «micros», dificultad de realización de los programas y mayor cantidad de instrucciones.

del microprocesador siendo mucho más trabajoso. Los programas son difíciles de realizar porque cuesta más memorizar una ristra de ceros y unos que unas palabras relacionadas con lo que queremos hacer (así 00111100 se olvida más fácilmente que **LET A=A+1**), pero todavía se complica más si decimos que en el código máquina hay más instrucciones que en un lenguaje de alto nivel.
Sin embargo podemos casi solucionar el problema de programar con ceros y unos si tuviéramos comandos que nos recordasen la función que realizan. Esto es lo que nos aporta el lenguaje ENSAMBLADOR.

EL ENSAMBLADOR

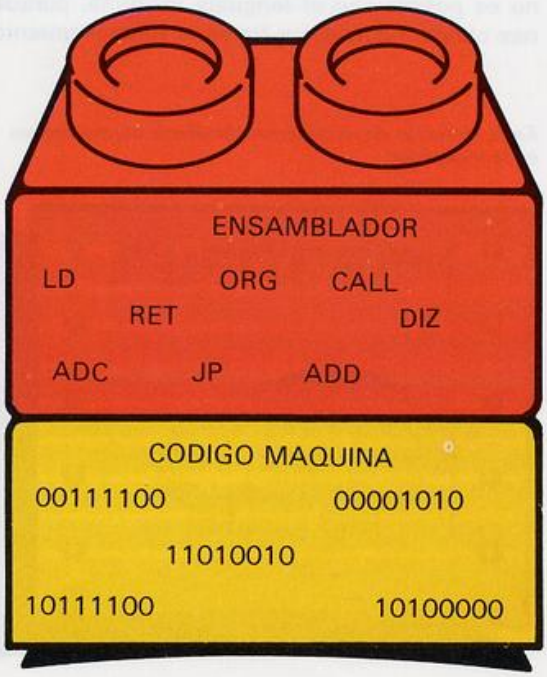
El ENSAMBLADOR es un lenguaje que tiene unos comandos que nos recuerdan qué función realizan y que se corresponden directamente con una instrucción del código máquina.

*

El ENSAMBLADOR sólo «sabe» realizar sumas y restas.

*

El ENSAMBLADOR es un lenguaje de ayuda a la programación en código máquina de bajo nivel porque cada mnemónico es equivalente a una instrucción en el código máquina.



Si a cada una de las 200 instrucciones del código máquina (formadas todas por ocho unos y ceros) las asociásemos un símbolo que nos recordarse la función que realizan y que fuera equivalente a cada comando del lenguaje máquina, nos costaría menos programar. Este conjunto de símbolos (que a partir de ahora llamaremos mnemónicos) es el que forma el lenguaje ENSAMBLADOR. Como último detalle, añadir que el mismo ENSAMBLADOR se encargará de realizar la traducción de los mnemónicos a los comandos del código máquina.
El ensamblador es un lenguaje de ayuda a la programación en código máquina de bajo nivel, porque cada comando, o mnemónico, es equivalente a una instrucción en el código máquina, y aunque sea necesaria una traducción, debido a la equivalencia que presenta apenas se nota una pérdida de velocidad.
El lenguaje ensamblador aporta las mismas ventajas e inconvenientes que el código máquina, salvo la facilidad de recordar los mnemónicos, que aunque no son los comandos de los lenguajes de alto nivel, son mucho mejores que las series de números del código máquina.

Entonces, ¿será siempre aconsejable programar en ENSAMBLADOR? Es evidente que aunque nos facilita el trabajo no todo es tan sencillo como en el BASIC, por lo tanto hay que pensarse si en nuestro programa es conveniente utilizarlo o no. El ENSAMBLADOR (y también el código máquina) es bueno para todo aquello que necesite velocidad, por ejemplo para los juegos de *arcade* (los típicos de masacrar marcianitos), para manejar pantallas (cambios de paisaje en un juego), trabajar con ficheros, al ser posible encontrar un dato de forma muy rápida, para instrucciones que se repitan muy frecuentemente; al igual que para todo lo que necesite ahorro de memoria, como en los programas largos o ficheros. Por el contrario, no es nada aconsejable para todas aquellas cosas simples, como la impresión de mensajes por pantalla, puesto que en estos casos no se aprecia ninguna ventaja al hacerlo por BASIC o ENSAMBLADOR.

No existe una fórmula mágica para decidir en cual lenguaje programar, aquí hemos pretendido dar unas ideas, pero es el programador el que tiene la última palabra dependiendo del tiempo disponible para trabajar (el proceso es más largo, evidentemente, en el ensamblador), de la memoria que disponga nuestra máquina, la velocidad requerida, las necesidades que pretendamos satisfacer y otros factores que determinemos. Sin embargo, no es necesario realizar un programa entero en ENSAMBLADOR dado que podemos combinarlo perfectamente con el BASIC, y ahorrarnos el esfuerzo de «ensamblar» aspectos no interesantes.

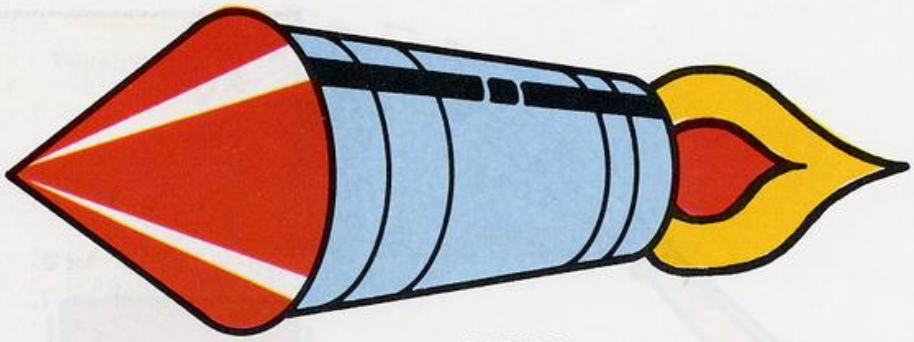
PROFUNDIZAMOS

Ahora ya nos animamos más a programar en código máquina, porque nosotros lo hacemos en ENSAMBLADOR y él mismo se encargará en pasarlo traducido a la máquina.

Sin embargo, para ello necesitamos que nuestra máquina tenga posibilidad de incorporarle un ensamblador. Eso no es problema para el SPECTRUM que dispone de muchas versiones. Pero para tener una idea general de cómo es este lenguaje nos basta la versión de *ARTIC COMPUTING* comercializada por INDESCOMP.

Antes de continuar adelante es necesario diferenciar lo que es un «programa ensamblador» o un «programa en ensamblador». Lo primero se refiere al lenguaje en sí, es decir al programa encargado de traducir los mnemónicos a código máquina. El segundo es el programa que nosotros confeccionaremos, que será posteriormente «ensamblado por el ENSAMBLADOR».

También debemos diferenciar entre lo que es el código fuente y el código objeto. El primero es el programa que escribimos lenguaje ensamblador; el segundo es el mismo programa traducido a lenguaje máquina por el ensamblador.



CODIGO MAQUINA

El lenguaje máquina es bastante más rápido que el BASIC.

En el código máquina tenemos más instrucciones que en un lenguaje de alto nivel.

CODIGO MAQUINA

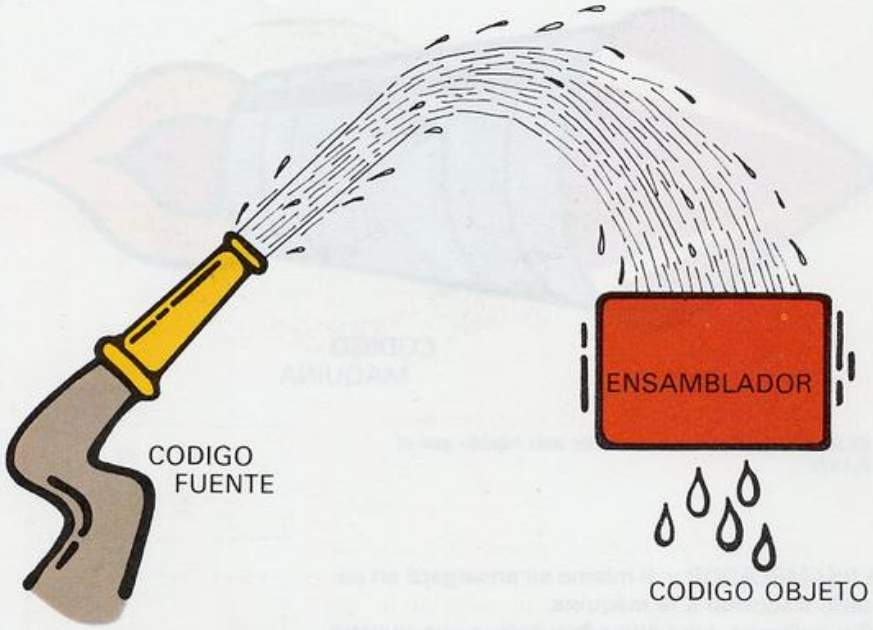
BASIC

¡!

Las ventajas que nos ofrece el código máquina es la de su velocidad, hasta 60 veces más rápido que el BASIC, y el ahorro de memoria.

*

El ENSAMBLADOR presenta las mismas ventajas e inconvenientes que el código máquina, salvo la facilidad de recordar los mnemónicos.



El código fuente es el programa escrito por nosotros, y el código objeto es cuando está traducido a lenguaje máquina.

En este ensamblador el código fuente es situado automáticamente en una línea **REM** con el número 2. El código objeto lo hace también en una **REM** en la línea 1, a no ser que voluntariamente decidamos cambiar su emplazamiento con **ORG**. Por tanto, para grabar ambos códigos se hace como si fuera un programa BASIC y no como *bytes*, que es lo que hacen otros ensambladores. Unos consejos importantes:

— En los programas largos grabar el código fuente y hacerlo por partes, para evitar disgustos cuando tengamos casi todo el programa tecleado (por ejemplo que nos quedemos sin luz).

— Antes de ejecutar un programa grabarlo. Si hay un error y se bloquea la máquina nos evitaremos teclear todo de nuevo.

— Cuando todo funcione ya se puede borrar el código fuente (**REM** línea 2 en el programa de **ARTIC**), aunque conviene guardar una copia de él para posibles rectificaciones posteriores.

Una de las diferencias entre el BASIC y el ENSAMBLADOR es que éste no utiliza variables. Sus «variables» son llamadas registros, y sirven para guardar la información provisionalmente. Es en ellas donde se realizan las operaciones aritméticas y lógicas. Por tanto, cada vez que se use la palabra registro debemos pensar en un espacio en la memoria para guardar la información.

Una vez cargado el ENSAMBLADOR estaremos directamente en el modo de inserción, pudiendo ya empezar a escribir nuestro programa. En caso de haber cometido algún error al escribirlo debemos entrar en el Editor para corregirlo. Para ello se pulsa **E** seguido de **ENTER** y tendremos acceso al menú que nos servirá para solucionar los problemas.

LOS PRIMEROS COMANDOS

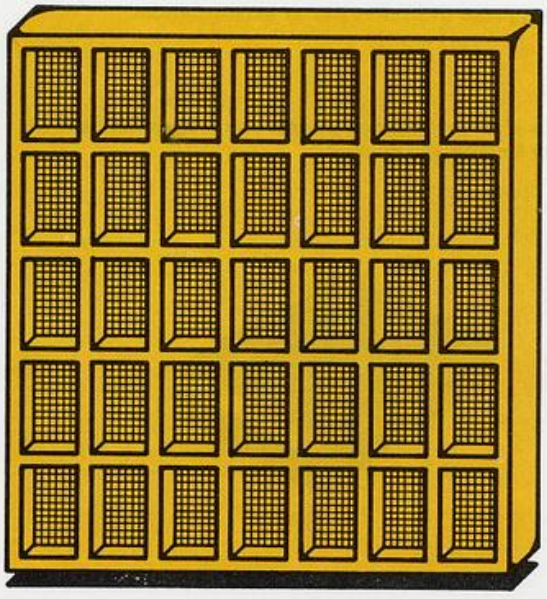
¡!

Un «programa ensamblador» es el encargado de traducir los mnemónicos a código máquina. Un «programa en ensamblador» es el que nosotros hacemos en este lenguaje.

*

Código fuente es el programa escrito por nosotros en ENSAMBLADOR. Código objeto es el mismo programa traducido (o ensamblado) a código máquina.

Registros es lo que usa el ENSAMBLADOR para guardar los datos.



Ya a estas alturas nos hemos podido percatar que el ensamblador es un lenguaje más, aunque tenga sus rarezas (todos los lenguajes las tienen). Los mnemónicos en ENSAMBLADOR realizan operaciones más simples, por eso es un lenguaje de bajo nivel, traducido directamente al código máquina, pero por el contrario ejecutable a una mayor velocidad. Otra particularidad que presentan es que tienen capacidad para manejar directamente direcciones de memoria, almacenando datos en donde consideremos oportuno, ante lo cual debemos tomar precauciones de dónde guardamos para que no se nos pierda nada por el intrincado bosque de las celdas de memoria.

El ensamblador de INVESTRONICA tiene la particularidad de preferir los números hexadecimales a los decimales. Para los primeros no hay que especificar nada, mientras que para los segundos se antepone el signo «+» o «–» para que sepa que ese número es decimal.

Una de las instrucciones más similares a los lenguajes de alto nivel es **LD**, cuya función es la de asignación (el **LET** del BASIC). Su forma es:

LD A,2

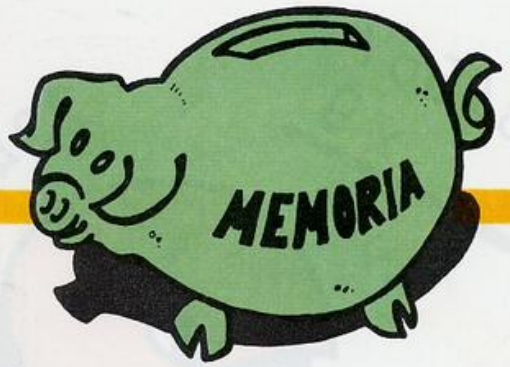
Aquí guardamos el valor «2» en el registro «A» (recordemos que no es lo mismo un registro que una variable, aunque, sin que se entere un informático, lo consideraremos equivalente).
Una variante del mismo comando es:

LD (HL),2

En esta ocasión guardamos el dato (recordemos que está en hexadecimal) dentro de la dirección de memoria indicada en el registro «HL». Es decir, en este registro tenemos almacenado un número el cual nos indicará el número de celda de memoria en que debemos guardar el dato que se encuentra a continuación. Creo que no ofrecerá ningún problema lo siguiente:

LD A,(HL)

¿Lo traducimos? Ahora guardaremos en el registro «A» el valor que se encuentra en la dirección de memoria indicada por «HL». Por tanto cada vez que utilicemos «()» se hará referencia a la dirección de memoria y no al valor almacenado.
La instrucción **CALL** supone un salto en la ejecución normal del programa, ya que éste se continuará ejecutando en la dirección de memoria in-



El lenguaje máquina presenta también la ventaja del ahorro de memoria.

Debemos analizar lo que vamos a programar para saber si resulta rentable hacerlo en ENSAMBLADOR.

¡!

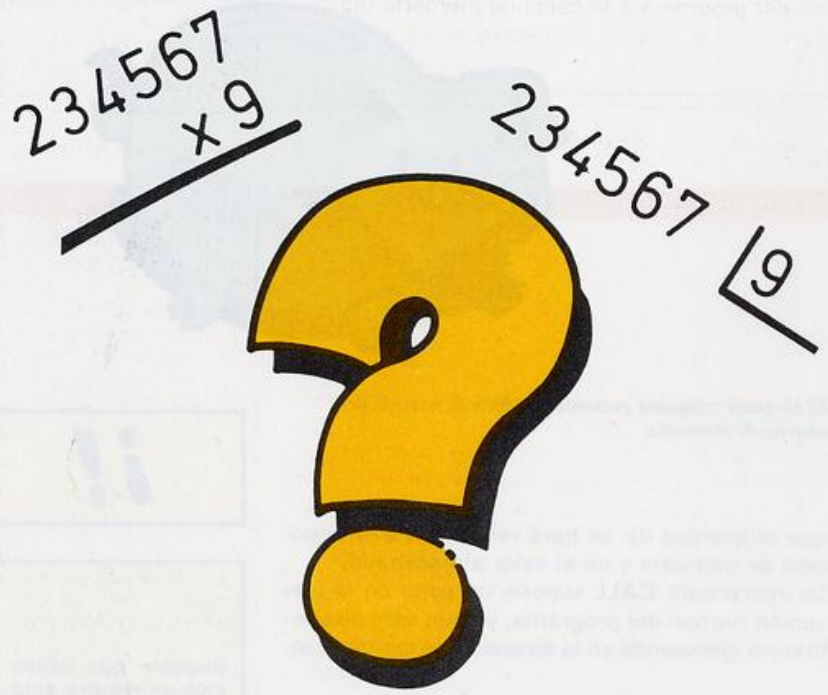
Siempre que utilicemos un registro entre «()» se refiere a la dirección de memoria indicada por el valor que tenga el registro dentro de «()».

*

Los mnemónicos realizan operaciones más simples pero más rápidas, y tienen la facultad de manejar directamente direcciones de memoria.

*

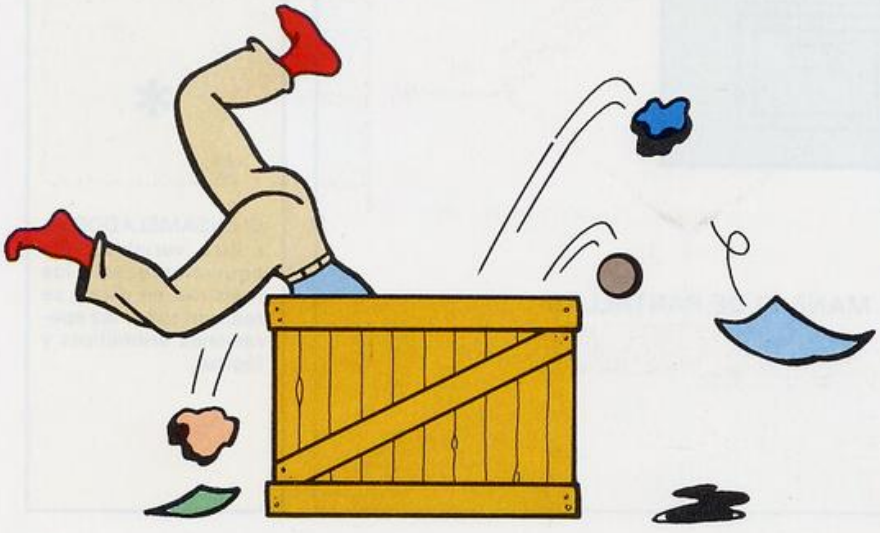
El ENSAMBLADOR no utiliza variables. Su equivalente son los registros, en donde se realizan todas las operaciones aritméticas y lógicas.



El ensamblador no «sabe» realizar otras operaciones diferentes a sumar y restar.

dicada a continuación. Nos sirve para, a partir de dicha dirección, colocar una subrutina. Tenemos también la orden contraria, **RET**, que retorna al punto al que ha sido llamado por una instrucción **CALL** o retorna al BASIC si está llamada se ha producido desde éste (con el comando **RANDOMIZE USR** ...), y, si es así, no nos debemos olvidar colocar este comando al final de nuestro programa en ensamblador para evitar que se nos quede «colgado» y no podamos recuperar el control del programa.

Debemos tener cuidado en qué celda de memoria hemos guardado los datos para recuperarlos cuando haga falta.



LA PILA

Es informática una pila es una forma de guardar provisionalmente unos datos numéricos. En el caso del ENSAMBLADOR se puede simular a esos archivadores que tienen un clavo largo donde se pinchan los papeles. Si clavamos varios papeles, el último que hayamos metido será el primero que podamos sacar (de una manera normal, se supone). A este tipo de pila se la conoce como *LIFO (Last In, First Out),* último dato en entrar, primero en salir.

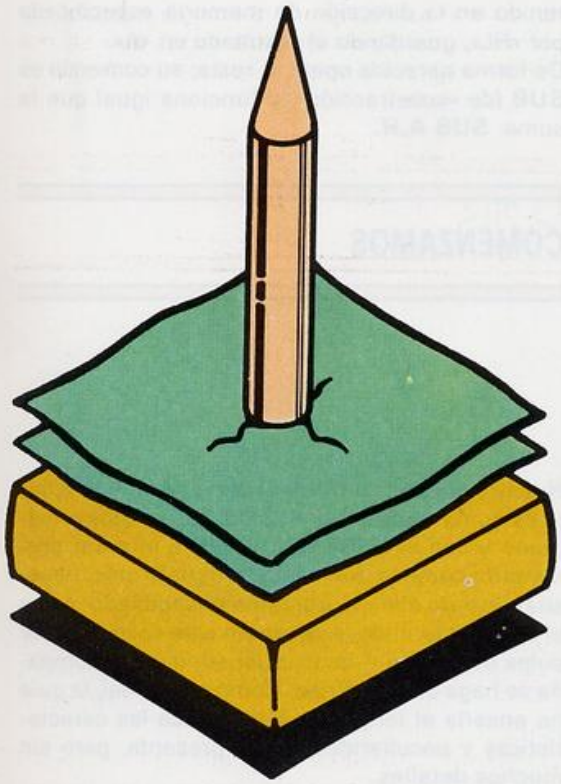
El ENSAMBLADOR utiliza la pila (también conocida como *stack*) para almacenar datos de una forma provisional no siendo posible realizar ninguna operación en ella. La única finalidad que tiene es guardar números cuando tengamos todos los registros ocupados.

Para trabajar con la pila tenemos dos comandos. El primero de ellos es **PUSH** («poner»), que no nos sorprenderá al conocer su misión: guardar la información en la pila. Así, **PUSH AF** almacenará en la pila el valor que se encuentre en el registro «AF».

Pero no nos serviría de nada si la información fuera irrecuperable. Para ello disponemos de **POP** (¿no nos suena a «sacar» el tapón de una botella de vino?). El mecanismo es igual que en la orden anterior: saca el valor de la pila, el superior (acuérdese del ejemplo del pincho de papeles, sólo podemos extraer el último introducido), y lo almacena en el registro indicado: **POP AF**.

INCREMENTAMOS Y DECREMENTAMOS

En ENSAMBLADOR disponemos de la facilidad de unos contadores que aumentarán o disminui-



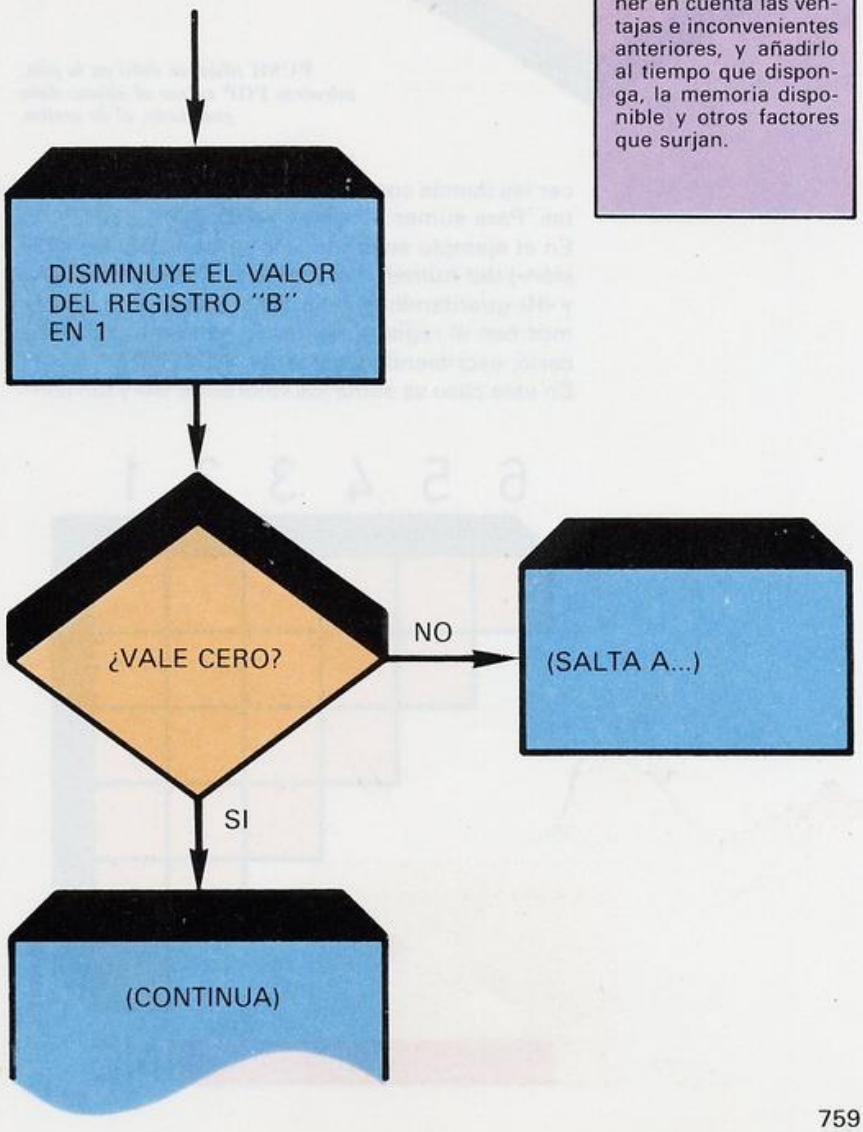
Una pila es un medio para guardar datos numéricos. El último dato en entrar en la pila es el primero en salir.

rán el valor de lo contenido en el registro al que se haga referencia en una unidad.
Aquí seguimos contando con la ventaja que aporta el ENSAMBLADOR al recordar la escritura de los comandos la función que realizan.
Así, **INC** «incrementa» en 1 el valor del dato que se encuentra en el registro indicado. Veamos el siguiente ejemplo: **INC HL**. El contenido del registro HL es aumentado en uno tras ejecutar este comando.
También podemos «decrementar» el valor de un registro en una unidad. La sintaxis y el significado no presentarán ningún problema: **DEC HL**.
Podemos complicar todavía más la situación con un salto condicional combinado con un decremento, y todo esto en un solo comando.
Para ello tenemos **DJNZ**, cuya función cada vez que se ejecuta es (¿preparados?) decrementar el valor que se encuentra en el registro «B» y si no es cero se ejecutará lo que se encuentre a continuación del comando, y si lo es no habrá salto y seguirá por las instrucciones que se encuentran debajo.
Tenemos también, como en todos los lenguajes, los saltos incondicionales. El representante más simple es **JP** (del inglés *jump*). Por ejemplo, **JP (HL)**. Acabamos de producir un salto en la ejecución del programa a la celda de memoria indicada por el valor que se encuentra en «HL».

SUMAMOS Y RESTAMOS

No nos llevaremos ninguna sorpresa si le decimos que con el ensamblador podemos sumar y restar. Aunque sí posiblemente si le decimos que sólo «sabe» realizar estas dos operaciones, y en realidad no necesita más, porque se pueden ha-

Aquí se ve la función de DJNZ.

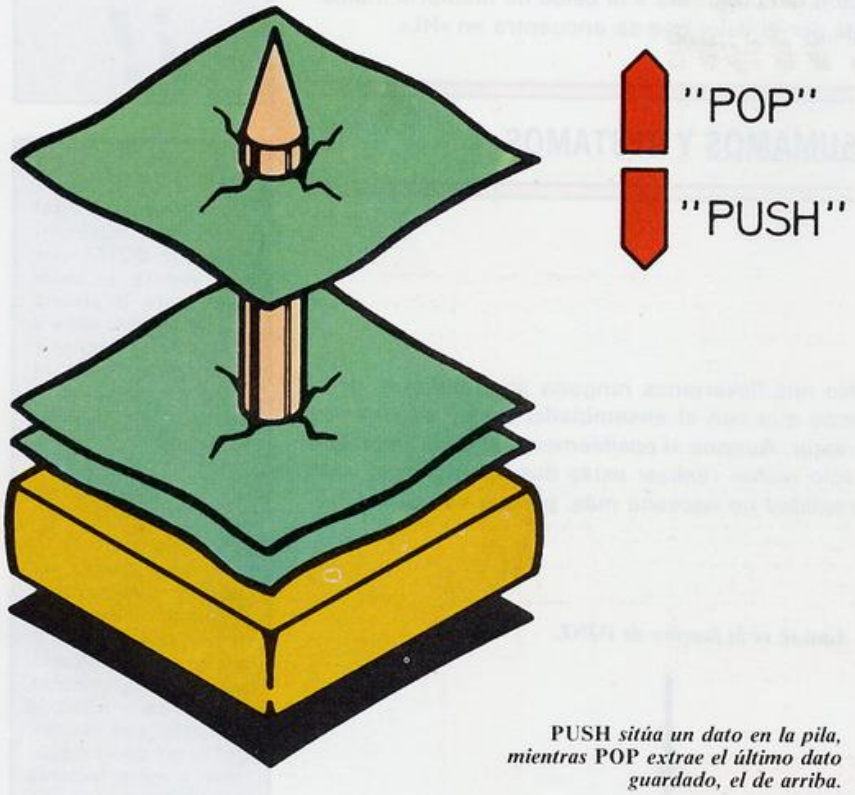


¡!

El salto condicional puede estar representado por **DJNZ** que decrementa el valor del registro «B» en uno y si no es cero salta a donde se le indiquen y si lo es continua la ejecución del programa.

*

La última palabra sobre cual lenguaje utilizar la tiene el programador, que deberá tener en cuenta las ventajas e inconvenientes anteriores, y añadirlo al tiempo que disponga, la memoria disponible y otros factores que surjan.



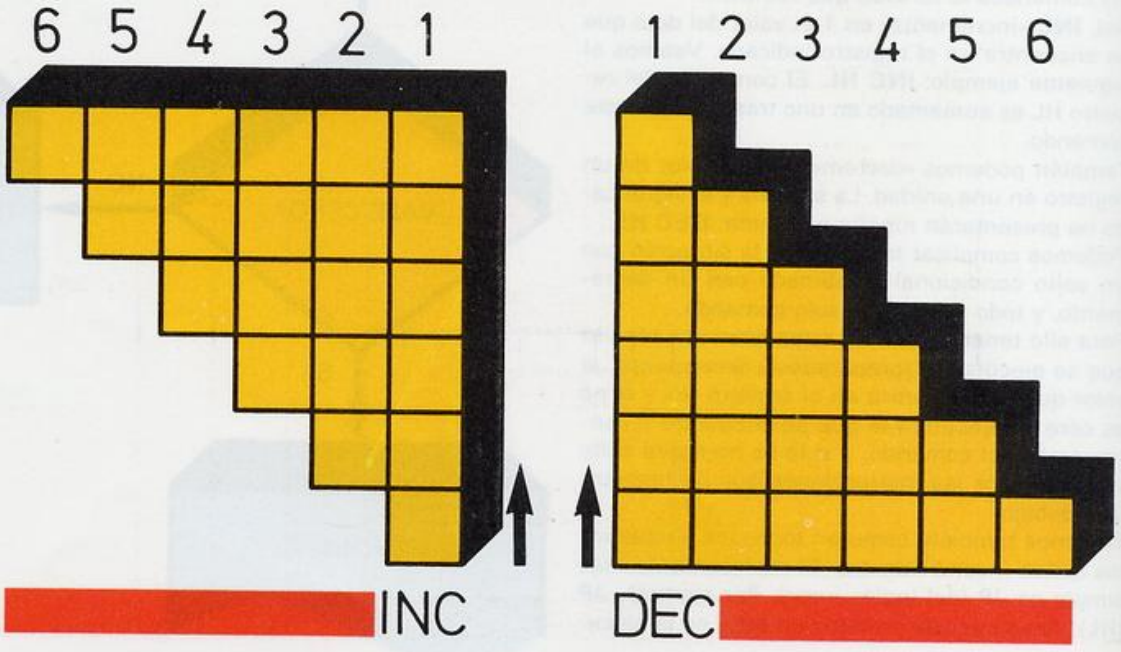
PUSH *sitúa un dato en la pila, mientras* **POP** *extrae el último dato guardado, el de arriba.*

cer las demás como combinación de sumas y restas. Para sumar haremos: **ADD A,H.**
En el ejemplo se produce la suma (**ADD** de «adición») del número contenido en los registros «A» y «H» guardando el valor en el primero. Si operamos con el registro «A» no es necesario especificarlo, escribiendo solamente: **ADD (HL).**
En este caso se suma los valores de «A» y del contenido en la dirección de memoria especificada por «HL», guardando el resultado en «A».
De forma parecida opera la resta; su comando es **SUB** (de «substracción», y funciona igual que la suma: **SUB A,H.**

COMENZAMOS

Nuestro objetivo al tratar el tema de este lenguaje es quitar el miedo que existe con el código máquina y con el ENSAMBLADOR, e intentar presentarlo como lo que es: un lenguaje más, necesitando para ello un programa ensamblador como el aquí comentado, a pesar, en este caso, que por culpa de las «mini-instrucciones» que le acompaña se haga cuesta arriba. Como es normal, la guía no enseña el lenguaje, sólo explica las características y peculiaridades que presenta, pero sin muchos detalles.
Esperemos que esa caja negra que es para muchos el ENSAMBLADOR, haya desaparecido, y se anime a hacer sus primeros pinitos, no conformándose con lo aquí expuesto, sino aprovechar este final para comenzar a «ensamblar».

INC *aumenta en uno el valor de lo contenido en el registro y* **DEC** *disminuye dicho valor en uno.*



MICROARITMETICA

La ALU (unidad aritmética y lógica) situada en el interior del Z 80 es la encargada de gestionar, entre otros trabajos, aquellos que impliquen la ejecución de sumas, restas y manejo de los operadores lógicos AND, OR y XOR.
Estos últimos ya fueron discutidos en el capítulo anterior, por lo que en este nos centraremos en las instrucciones aritméticas previstas por los fabricantes del microprocesador instalado en nuestro Spectrum.

Grupo aritmético de 16 bits.

MNEMONICO	CODIGO MAQUINA	REGISTRO F 7 S	6 Z	5	4 H	3	2 P/V	1 N	0 C	Nº BYTES	CICLOS MAQ.	CICLOS RELOJ	NOTAS
ADD HL, ss	o o s s 1 o o 1	•	•		x		•	0	↕	1	3	11	
ADC HL, ss	1 1 1 o 1 1 o 1 o 1 s s 1 o 1 o	↕	↕		x		v	0	↕	2	4	15	
SBC HL, ss	1 1 1 o 1 1 o 1 o 1 s s o o 1 o	↕	↕		x		v	1	↕	2	4	15	
ADD IX, pp	1 1 o 1 1 1 o 1 o o p p 1 o o 1	•	•		x		•	0	↕	2	4	15	
ADD IY, rr	1 1 1 1 1 1 o 1 o o r r 1 o o 1	•	•		x		•	0	↕	2	4	15	
INC ss	o o s s o o 1 1	•	•		•		•	•	•	1	1	6	
INC IX	1 1 o 1 1 1 o 1 o o 1 o o o 1 1	•	•		•		•	•	•	2	2	10	
INC IY	1 1 1 1 1 1 o 1 o o 1 o o o 1 1	•	•		•		•	•	•	2	2	10	
DEC ss	o o s s 1 o 1 1	•	•		•		•	•	•	1	1	6	
DEC IX	1 1 o 1 1 1 o 1 o o 1 o 1 o 1 1	•	•		•		•	•	•	2	2	10	
DEC IY	1 1 1 1 1 1 o 1 o o 1 o 1 o 1 1	•	•		•		•	•	•	2	2	10	

Reg.	ss
BC	00
DE	01
HL	10
SP	11

Reg.	PP
BC	00
DE	01
IX	10
SP	11

Reg.	rr
BC	00
DE	01
IY	10
SP	11

GRUPO ARITMETICO DE 8 BITS

Pertenecen a este grupo, todas aquellas instrucciones que efectúen la suma (ADD) o resta (SUB) de un valor contenido en un registro o posición de memoria con el almacenado en el acumulador, quedando el resultado en este último y manteniéndose inalterado el valor del registro o posición de memoria implicada tras la operación aritmética.

Genéricamente podríamos representar estas operaciones de la manera siguiente:

SUMA A+n->A
RESTA A-n->A

MNEMONICO	CODIGO MAQUINA	REGISTRO F 7 S	6 Z	5	4 H	3	2 P/V	1 N	0 C	Nº BYTES	CICLOS MAQ.	CICLOS RELOJ	NOTAS
ADD r	1 0 0 0 0 r r r	↕	↕		↕		v	0	↕	1	1	4	
ADD n	1 1 0 0 0 1 1 0 n n n n n n n n	↕	↕		↕		v	0	↕	2	2	7	
ADD (HL)	1 0 0 0 0 1 1 0	↕	↕		↕		v	0	↕	1	1	4	
ADD (IX+d)	1 1 0 1 1 1 0 1 1 0 0 0 0 1 1 0 d d d d d d d d	↕	↕		↕		v	0	↕	3	5	19	
ADD (IY+d)	1 1 1 1 1 1 0 1 1 0 0 0 0 1 1 0 d d d d d d d d	↕	↕		↕		v	0	↕	3	5	19	
ADC s	0 0 1	↕	↕		↕		v	0	↕	(a)	(b)	(c)	
SUB s	0 1 0	↕	↕		↕		v	1	↕				
SBC s	0 1 1	↕	↕		↕		v	1	↕				
INC r	0 0 r r r 1 0 0	↕	↕		↕		v	0	•	1	1	4	
INC (HL)	0 0 1 1 0 1 0 0	↕	↕		↕		v	0	•	1	3	11	
INC (IX+d)	1 1 0 1 1 1 0 1 0 0 1 1 0 1 0 0 d d d d d d d d	↕	↕		↕		v	0	•	3	6	23	
DEC s	1 1 1 1 1 1 0 1 0 0 1 1 0 1 0 1 d d d d d d d d	↕	↕		↕		v	1	•	3	6	23	

Reg.	
A	111
B	000
C	001
D	010
E	011
H	100
L	101

S
r
n*
(HL)
(IX+d)
(IY+d)

* Salvo en DECs

- • Indicador no afectado. Indicador modificado según el resultado de la operación.
- P El bit 2 del registro F actúa como indicador de pondad (1 – pondad par/0 – pondad impar).
- V El bit 2 del registro F actúa como indicador de sobrepasamiento (1 – sobrepasam./ 0 – no sobrepasam.).
- X Estado indeterminado.

(a) (b) (c) En las instrucciones donde se utiliza la abreviatura "s" el nº de bytes, ciclos máquina y de reloj, son los mismos que sus correspondientes anteriores.

Grupo aritmético de 8 bits.

donde A representa el valor almacenado en el acumulador, y n el contenido del registro o posición de memoria especificado en la instrucción. Además, es posible incrementar (INC) o decrementar (DEC) en 1 la cantidad almacenada en un determinado registro. Veamos el efecto que sobre los indicadores tienen estas operaciones:

S.—Cuando el resultado en A es negativo, es decir, si el bit más significativo de éste es 1, la bandera de signo contendrá un 1. En caso contrario será 0.

Z.—Tras una operación aritmética, su valor será 1, si el resultado de ésta fue 0. De lo contrario, contendrá un 0.

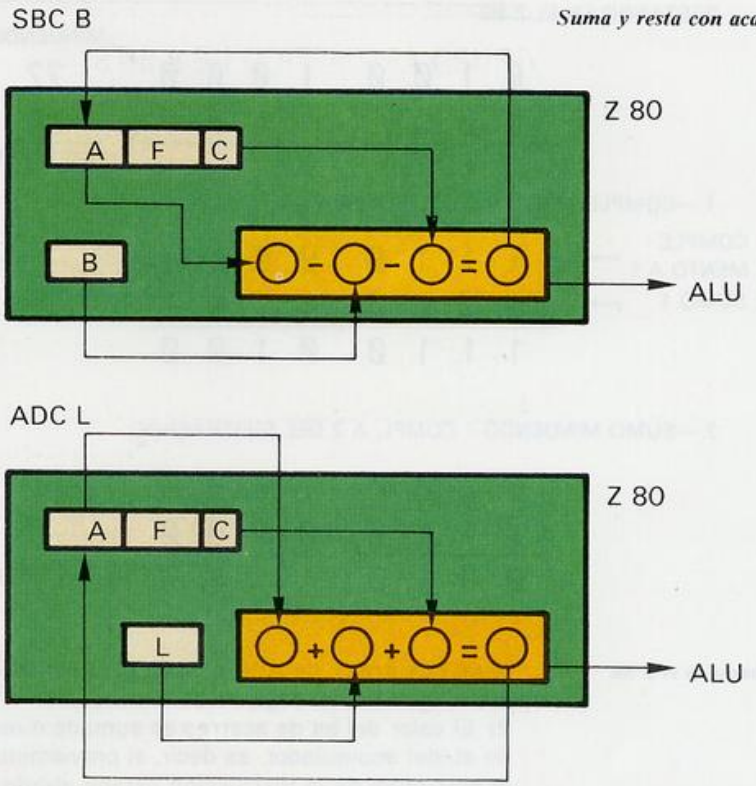
N.—El indicador de resta se coloca a 1 en todas las operaciones relacionadas con esta operación, mientras que en las sumas permanece a 0.

P/V.—Señala si hubo sobrepasamiento (1) o no (0) en las operaciones aritméticas de dos números expresados en complemento a dos.

C.—Actúa de manera similar al anterior, pero con números expresados en su correspondiente formato binario.

H.—Su importancia se pone de manifiesto cuando ejecutamos operaciones aritméticas con números en formato BCD (decimal codificado en binario), los cuales serán discutidos próximamente, al analizar la instrucción DAA.

En el capítulo dedicado a la presentación de las tablas de instrucciones, se señalan los operandos sobre los cuales actúan las de mnemónico ADD, SUB, INC y DEC. Como podemos comprobar, además están contempladas dos operaciones especiales para la suma y la resta, las cuales tienen presente el valor de la bandera de arrastre C. Se trata de aquellas cuyo mnemónico es ADC (sumar con acarreo) y SBC (restar con acarreo).



Suma y resta con acarreo.

Implementar estas instrucciones obliga al microprocesador a seguir los siguientes pasos basados en el esquema:

SUMA con acarreo A+n+C->A

RESTA con acarreo A-n-C->A

1) Suma o resta al/del acumulador el byte especificado en la instrucción como si se tratara, al

MNEMONIACO	CODIGO MAQUINA	REGISTRO F 7 S	6 Z	5	4 H	3	2 P/V	1 N	0 C	Nº BYTES	CICLOS MAQ.	CICLOS RELOJ	NOTAS
DAA	00100111	↕	↕		↕		P	•	↕	1	1	4	
CPL	00101111	•	•		1		•	1	•	1	1	4	
NEG	11101101 01000100	↕	↕		↕		v	1	↕	2	2	8	
CCF	00111111	•	•		x		•	0	↕	1	1	4	
SCF	00110111	•	•		0		•	0	1	1	1	4	
NOP	00000000	•	•		•		•	•	•	1	1	4	
HALT	01110110	•	•		•		•	•	•	1	1	4	

Grupo aritmético de la C.P.U.

RESTANDO EN EL Z-80

```
                                   MINUENDO
  Ø 1 Ø Ø  1 Ø Ø Ø    72
 -Ø Ø Ø 1  1 1 Ø Ø   -28
                     SUSTRAENDO
```

1.—COMPLEMENTO A 2 DEL SUSTRAENDO

```
1.1. COMPLEMENTO A 1 →  1 1 1 Ø  Ø Ø 1 1
1.2. SUMO 1         → +Ø Ø Ø Ø  Ø Ø Ø 1
                      1 1 1 Ø  Ø 1 Ø Ø
```

2.—SUMO MINUENDO + COMPL. A 2 DEL SUSTRAENDO

```
  Ø 1 Ø Ø  1 Ø Ø Ø
 +1 1 1 Ø  Ø 1 Ø Ø
  Ø Ø 1 Ø  1 1 Ø Ø  → 44 DECIMAL
```

Restando en el Z 80.

igual que antes, de ADD o SUB, pero sin modificar ningún bit del registro de indicadores.

2) El valor del bit de acarreo es sumado o restado al/del acumulador, es decir, si previamente a la ejecución de la instrucción estaba alzado (1), se suma o resta 1 de A.

3) Finalmente el Z 80 ajusta las banderas del registro F, basándose en el último resultado recogido en A.

¡!

El BCD (Binary Coded Decimal) es un sistema para codificar números decimales empleando 4 dígitos binarios por cada uno decimal.

*

Cuando trabajamos en aritmética BCD, tras cualquier operación debemos ejecutar una instrucción DAA para evitar posibles confusiones.

EL GRUPO ARITMETICO DE 16 BITS

Las instrucciones aritméticas anteriormente comentadas ADD, ADC, SBC, INC y DEC, que actúan sobre registros o posiciones de memoria de 8 bits, tienen sus análogas cuando se trata de operar con 16 bits, pero solamente trabajan con cantidades almacenadas en registros dobles, no siendo posible recogerlas de una celda de memoria determinada.

En el caso de las sumas y restas con 8 bits, el primer operando no era necesario especificarlo en la instrucción, pues el microprocesador tomaba siempre por defecto el valor almacenado en el acumulador.

Cuando se trata de sumar o restar números de 2 bytes, el primer operando ha de estar contenido en el par HL, salvo en las instrucciones ADD IX,pp y ADD IY,rr donde se toman los registros índice con este fin.

El segundo operando será el valor almacenado en cualquier par de registros de los autorizados en la instrucción en concreto de que se trate (ss, pp o rr), siguiéndose la decodificación expuesta en la tabla, sin que sea posible trabajar con parejas como BL o HC.

Tras efectuar la suma o resta, el resultado queda almacenado en HL (en IX o IY si es su caso), manteniéndose inalterado el contenido del par de registros utilizado como segundo operando.

Aclaremos su funcionamiento con algunos ejemplos: supongamos que deseamos ejecutar la instrucción ADD HL,DE siendo el valor contenido en HL 52277 y en DE 23758:

	ANTES	DESPUES
H	11001100	00101001
L	00110101	00000011
D	01011100	01011100
E	11001110	11001110

Es decir, tras sumar 52277 y 23758 ¡hemos obtenido como resultado 10499! en vez de 76035 como sería de esperar. Podríamos pensar que el Z 80 no tiene ni idea de lo que es una suma. Si nos fijamos en la tabla donde se detallan los indicadores afectados observaremos que sólo tres de ellos pueden haber cambiado.

La bandera de semiacarreo (H) se alzará en estas operaciones, cuando al ajecutar la suma binaria nos vayamos «llevando una» desde el bit 11 hasta el final. En nuestro ejemplo no ocurre esto y por tanto, permanecerá a 0. El indicador de resta N contendrá 1 cuando efectuemos, por ejemplo, una instrucción SBC, es decir, una resta. En nuestro caso la operación implementada fue una suma y contendrá 0, por tanto.

Hasta ahora no hemos conseguido detectar ninguna condición que permita justificar tan sorprendente resultado y sólo nos queda uno de los bits del registro F: el representativo del acarreo. Esta bandera se alza cuando se produce acarreo en el bit 15 (si me «llevo una» a partir del bit 15 al realizar la suma). Y eso precisamente es lo que ha ocurrido en nuestro ejemplo (comprobémoslo). Entonces el bit de acarreo C del registro F contendrá 1.

¿Cómo interpretamos entonces el resultado? Todos sabemos que con 16 bits podemos representar números decimales entre 0 y 65535, es decir 65536 combinaciones distintas. Pues 10499 es exactamente la cantidad resultante de efectuar la operación 52277+23758−65536.

Como el par HL sólo puede contener números de 16 bits, es decir, inferiores a 65535, el 76035 se le hace demasiado grande y por ello le resta 65536, circunstancia que detectamos al comprobar que el banderín de acarreo está alzado.

Las instrucciones de incremento y decremento de 16 bits son completamente análogas en su funcionamiento a sus correspondientes de 8 bits, sólo que ahora actúan sobre registros dobles.

Al ejecutarlas se suma/resta 1 del contenido del byte de menor peso dentro del par especificado en la instrucción. Si esta operación provoca acarreo, éste es transmitido automáticamente al octeto superior.

En los programas en C/M se utilizan para realizar bucles de más de 256 repeticiones, donde puede resultar interesante conocer cuando hemos llegado, a base de decrementar un valor prefijado, a 0.

Como estas instrucciones no afectan al contenido del registro de indicadores podríamos pensar que la anterior circunstancia no es detectable. Sin embargo, podemos servirnos del siguiente artificio: supongamos que hemos estado decrementando el par BC. Tras ejecutar las instrucciones

```
LD A,B
OR C
```

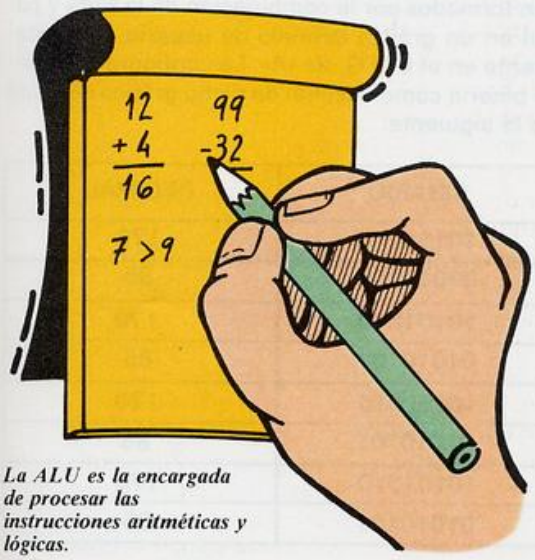
la bandera de cero **Z** contendrá un 1, solamente cuando tanto B como C sean 0.

EL GRUPO ARITMETICO DE LA CPU

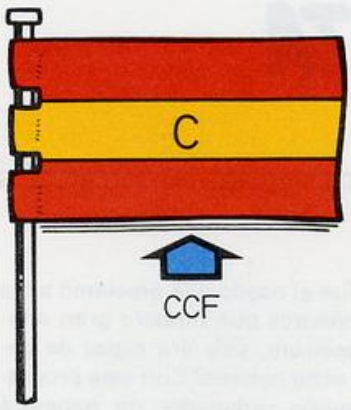
Pertenecen a éste algunas de las instrucciones incluidas en el capítulo 41 bajo la denominación «aritmético y de control de la CPU», las cuales vienen a complementar a las anteriores. Comentemos cual es exactamente su cometido:

— CPL (ComPLement, complementar) efectúa el complemento a 1 del valor almacenado en el acumulador, es decir, cambia los 1 por 0 y los 0 por 1.

— NEG (NEGative, negativo) realiza la negación del acumulador, operación que ha de entenderse como encontrar su complemento a 2. En realidad, el microprocesador lo que hace es restar de 0 el contenido de A (0-A), y para ello, como en todas las operaciones de resta complementa a 2 el sustraendo y le suma después el minuendo.



La ALU es la encargada de procesar las instrucciones aritméticas y lógicas.



CCF complementa la bandera de acarreo.

— CCF (Complement Carry Flag) complementa el valor actual de la bandera de acarreo C, es decir, si contenía 1, la coloca 0 y viceversa.

— SCF (Set Carry Flag) pone a 1 el indicador de acarreo C.

— DAA (Decimal Adjustment Accumulator) se encarga de efectuar el ajuste decimal del acumulador tras una operación aritmética de dos números en formato BCD.

El BCD es un sistema de codificación encargado de representar cada dígito correspondiente a un número decimal mediante cuatro dígitos binarios. Por ejemplo, el número decimal 76 tendría el siguiente aspecto en BCD:

01110110

Si quisiéramos representar un número mayor, por ejemplo, 6592 necesitaríamos dos octetos: 01000101 y 10010010.

Al realizar una operación aritmética con números en este formato, lo normal es que el resultado recogido en el acumulador no esté en BCD y para transformarlo a éste, se utiliza la instrucción DAA, de manera que no se produzcan errores de interpretación entre cantidades en binario y BCD.

A la anteriores podemos añadir dos instrucciones, las cuales afectan directamente a la CPU. Se trata de NOP (No OPeration) la cual simplemente indica al Z 80 que no haga nada. Esto no quiere decir que ejecutar esta instrucción no tome tiempo, pues de hecho se requieren 4 ciclos de reloj para llevarla a cabo. Su utilidad versa en proporcionar espacio dentro de un programa al objeto de añadir luego otras instrucciones.

Finalmente la orden HALT provoca que el microprocesador pare cualquier trabajo que esté efectuando.

Con esto cerramos el análisis de los grupos aritmético y lógico del Z 80 emprendido en el capítulo anterior. En el próximo discutiremos las instrucciones que provocan bifurcaciones dentro de la secuencia de un programa.

La instrucción NOP precisa de 4 ciclos de reloj para completar su ejecución.

*

Las operaciones aritméticas entre números de 16 bits, sólo es posible efectuarlas cuando estos se encuentran almacenados en registros dobles.

*

El resultado de una operación aritmética de 8 bits siempre queda almacenado en el acumulador.

*

HALT provoca la detención en cualquier trabajo que el microprocesador esté efectuando.

PALETA

¡!

Para la grabación del programa emplearemos el comando **SAVE "PALETA"**, o bien, **SAVE "PALETA" LINE 10**, si deseamos que éste se ejecute automáticamente al finalizar su carga.

*

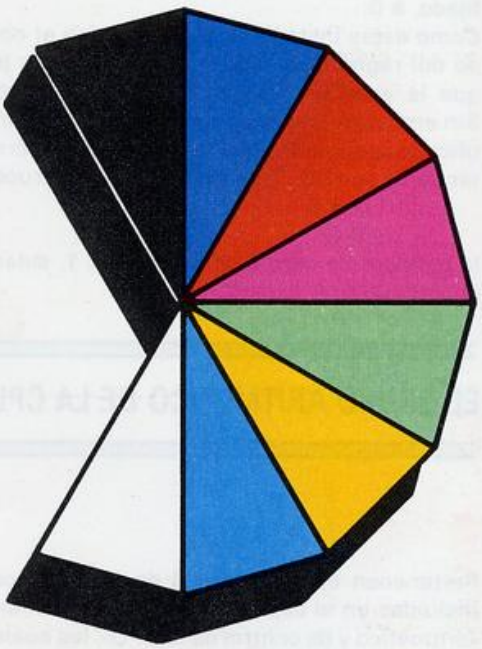
Los caracteres **A** subrayados en las líneas 1740 y 1920 del listado, corresponden al gráfico definido de dicha tecla.

Quién fue el osado que proclamó a los siete vientos que nuestro gran amigo Spectrum, sólo era capaz de generar ocho colores? Con este programa, nuestro pequeño ordenador no generará ocho, ni cuarenta, ni ochenta colores, sino ciento veintisiete. ¿Asombrados?

EL PROGRAMA

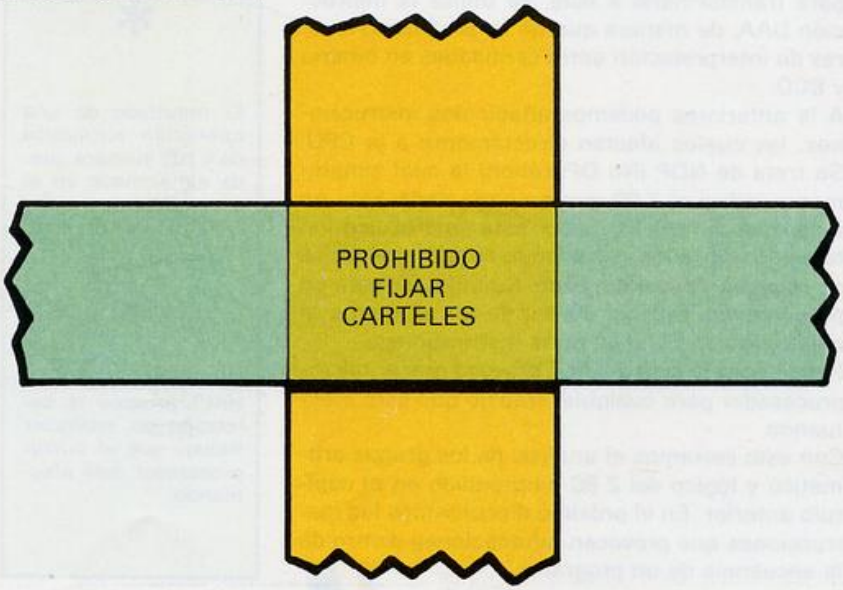
Nuestro programa PALETA genera ciento diecinueve mezclas, partiendo de los ocho colores básicos con los que normalmente puede trabajar el Spectrum. Estos ciento diecinueve tonos, junto con los ocho anteriormente mencionados forman el conjunto de ciento veintisiete colores.

Tanto los tonos primarios como las mezclas, están formados por la combinación de la tinta y papel en un gráfico definido de usuario, concretamente en el U.D.G. de «A». La configuración tanto binaria como decimal de dicho gráfico definido es la siguiente:



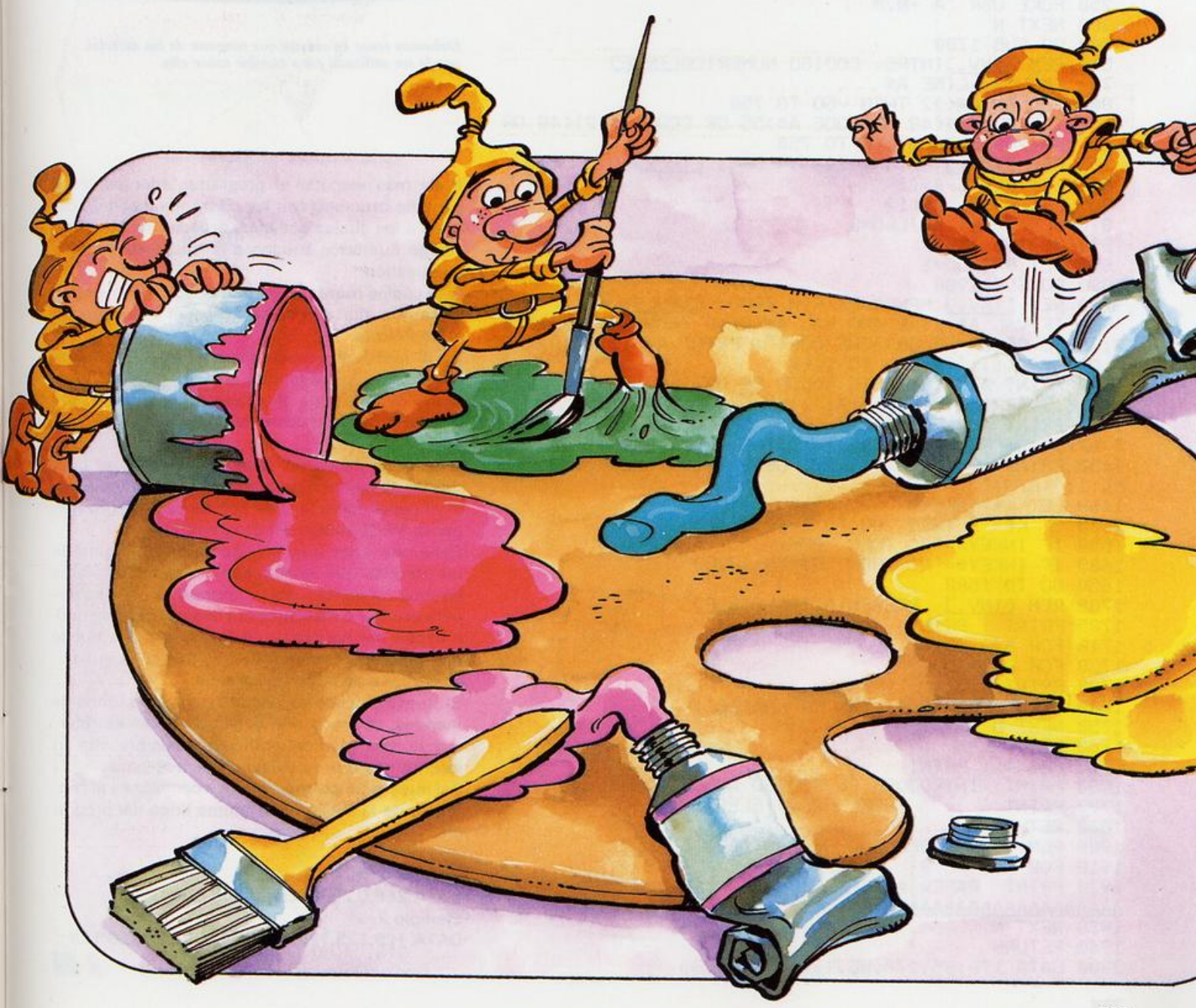
Nuestro programa PALETA genera 119 mezclas, partiendo de los 8 colores básicos del Spectrum.

Al ejecutar el programa aparece en la pantalla un menú con los diferentes tipos de colores, a los cuales podremos acceder mediante un código numérico.



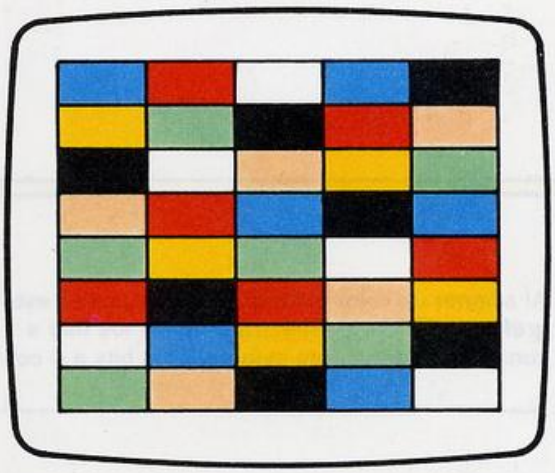
BINARIO	DECIMAL
10101010	170
01010101	85
10101010	170
01010101	85
10101010	170
01010101	85
10101010	170
01010101	85

Al asignar un color de tinta y otro de papel a este gráfico definido, el Spectrum activa los bits a 1 con el color de la tinta elegida, y los bits a 0 con el color del papel. Para nuestro ordenador sólo existirán dos colores, pero nosotros, al poseer una percepción óptica más o menos compleja, fu-



sionamos los dos tonos en uno solo, captando así la mezcla de los dos colores.

```
  10 REM ************************
  20 REM * J.M.MAYORAL SERRANO *
  30 REM ************************
  40 REM * PALETA      (C) 1986 *
  45 REM ************************
  50 POKE 23658,8
 100 PAPER 7: INK 9: BORDER 7
 150 CLS : PRINT  INK 1;"     127 COLORES PARA SPECTRU
M"
 160 REM [INV.] GRAFICOS DE USUARIO [TRUE]
 200 FOR N=0 TO 7
 210 READ M
 250 POKE USR "A"+N,M
 300 NEXT N
 500 GO SUB 1700
 510 REM [INV.]INTRO. CODIGO NUMERICO[TRUE]
 750 INPUT , LINE A$
 800 IF LEN A$<>2 THEN  GO TO 750
 850 IF CODE A$<48 OR CODE A$>55 OR CODE A$(2)<48 OR
CODE A$(2)>55 THEN  GO TO 750
 855 REM [INV.] ASIGNACION VALORES [TRUE]
 860 LET B=VAL  A$(2)
 865 LET A=VAL  A$(1)
 870 CLS : LET BRILLO=0
 880 GO SUB 1900
 890 LET BRILLO=1
 895 GO SUB 1900
 900 REM [INV.] MENSAJES FIN PROG. [TRUE]
 960 IF VAL A$(1)=0 AND VAL A$(2)=0 THEN  PRINT AT 8,
3;"ESTE COLOR SE OBTIENE CON:": PRINT AT 11,10;" EL C
OLOR ";A$(1);" ": GO TO 1100
1000 PRINT AT 3,3;"ESTE COLOR SE OBTIENE CON:": PRINT
 AT 5,2;"PRINT PAPER ";A;";INK ";B;";BRIGHT 0"
1010 PRINT AT 13,3;"ESTE COLOR SE OBTIENE CON:": PRIN
T AT 15,2;"PRINT PAPER ";A;";INK ";B;";BRIGHT 1"
1450 PRINT  INK 2;AT 21,2;" DESEAS CONTINUAR ?   "; I
NK 1;"(S/N)"
1460 LET K$=INKEY$
1500 IF K$="" THEN  GO TO 1460
1550 IF INKEY$="S" THEN  RUN
1600 IF INKEY$="N" THEN  GO TO 10000
1650 GO TO 1500
1700 REM [INV.]  PRESENTACION [TRUE]
1705 PRINT : PRINT
1710 FOR A=0 TO 7
1720 FOR B=0 TO 7
1730 PRINT A;B;
1740 PRINT  PAPER A; INK B;"AAAAAA";
1750 NEXT B
1760 NEXT A
1770 PRINT : PRINT
1780 PRINT "INTRODUCE EL CODIGO NUMERICO DEL"
1790 PRINT "        DEL COLOR ELEGIDO"
1800 RETURN
1900 REM [INV.] RELLENO DE PANTALLA [TRUE]
1910 FOR N=0 TO 9
1920 PRINT  PAPER A; INK B; BRIGHT BRILLO;"AAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAA"
1930 NEXT N
1940 RETURN
2000 DATA 170,85,170,85,170,85,170,85
```



Debemos tener en cuenta que ninguna de las mezclas puede ser utilizada para escribir sobre ella.

Nada más ejecutar el programa, aparece en la pantalla un menú con los diferentes tipos de colores a los cuales podemos acceder mediante un código numérico situado a la izquierda del tono en cuestión.

Este código numérico consta de dos cifras. La primera nos indica el color del papel y la segunda el de la tinta.

En el menú de presentación existen solamente 64 colores. Por cada tono elegido, nuestro Spectrum genera dos diferentes: uno sin brillo (**BRIGHT** 0) y otro con el brillo activado (**BRIGHT** 1).

Solamente existe un caso en donde el atributo «brillo» no ejerce ninguna influencia: cuando tanto el papel como la tinta son de color negro (paper 0; ink 0; código del programa=00).

Debemos tener muy en cuenta que ninguna de las mezclas que se obtienen de los ocho colores básicos no pueden ser utilizadas como «pizarra» para escribir o mensajes encima de estos, ya que, a causa del hardware, el Spectrum sólo puede mantener a un mismo tiempo dos colores básicos en una posición de caracter.

Si nuestro deseo es encontrar nuevos tonos de mezclas, lo único que debemos hacer es reformar la definición del gráfico definido por otra diferente a la que ya existe en el programa.

Así pues lo único que debemos cambiar es el contenido de la DATA en la última línea del programa.

Ejemplo 1.
DATA 255,0,255,0,255,0,255,0
Ejemplo 2.
DATA 129,129,129,129,129,129,129,129