47
185 pts.
IVA INCLUIDO

RUN

Enciclopedia Práctica del Spectrum

Nueva Lente/Ingelek

PASCAL

El PASCAL es actualmente el más firme candidato para desbancar al BASIC del lugar que ocupa dentro de los «micros», puesto que por su potencia, el PASCAL no sólo es apropiado para aprender informática sino para adentrarnos en cualquier campo sin caer en los defectos del BASIC.

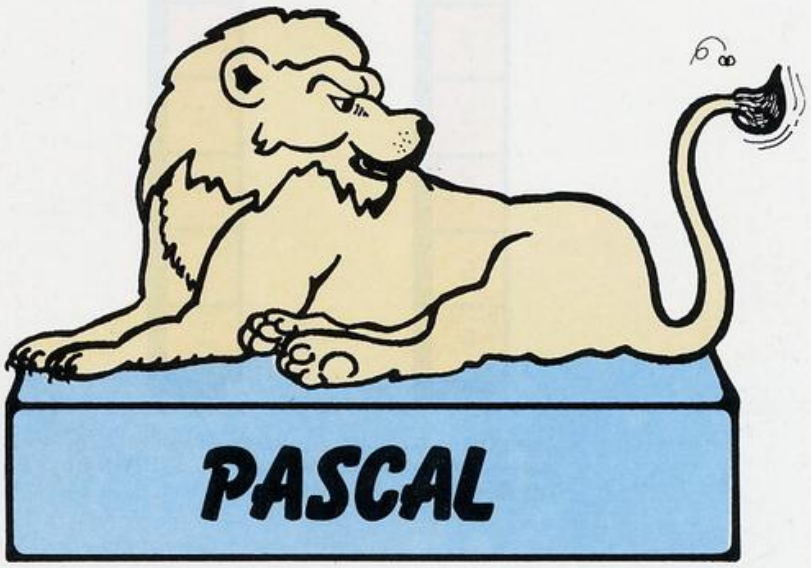
¿Por qué PASCAL? Este lenguaje surgió a finales de los años sesenta en un momento en que los ya existentes presentaban la particularidad que cuanto más largos eran los programas, más difícil era realizar un seguimiento visual de lo que hacen además de los problemas para realizar correcciones posteriores. En el caso particular del BASIC, el uso y abuso de la sentencia **GO TO** hace que para analizar un programa estemos saltando continuamente de un punto a otro disminuyendo bastante la inteligibilidad, pero que al ser un comando muy cómodo es fácil abusar de él.

El PASCAL soluciona estos problemas. Debido a su modularidad y a la posibilidad de trabajar con procedimientos, facilita bastante la comprensión y la corrección posterior de los programas.

Otro factor interviniente en el éxito que está teniendo el PASCAL, además de los ya mencionados, es que ocupa poca memoria y se puede incorporar en todos los microordenadores. Así ocurre con el ZX Spectrum que dispone de su propia versión PASCAL realizada por *HISOFT*. Es una versión que presenta como inconvenientes principales, además de tener la guía en inglés (pero que no sirve para nada a no ser que se conozca previamente el lenguaje), no presentar uno de los comandos para ficheros **(FILE)** que le da una gran potencia para el manejo de la información. No obstante, esta ausencia está motivada por una imposibilidad física, dado que no es posible soportarla en cinta, se hace imprescindible la utilización de un disco.

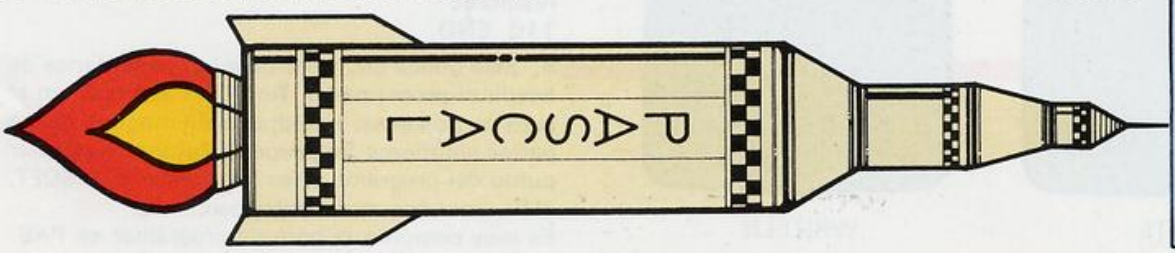
No debemos olvidar que el PASCAL es un lenguaje compilado, y que aparte de lo exigentes que son los compiladores (aunque en este caso es fácil acostumbrarse a ello) aporta la ventaja de su elevada velocidad, en ocasiones cuatro veces más rápido que el BASIC, lo cual hace su uso rentable para casi cualquier aplicación.

Una última advertencia: no nos desesperemos. El porqué se encuentra en que el BASIC crea unos vicios de programación (el abuso de **GO TO** y la no estructuración de los programas) que son incompatibles con la filosofía del PASCAL. Le costará tiempo y programas sacar el máximo provecho a este lenguaje pero una vez que lo consiga se dará cuenta de lo ventajoso que resulta. Debe tener en cuenta que el PASCAL ha sido creado



El PASCAL debido a su potencia y modularidad es uno de los más firmes candidatos para desplazar al BASIC en los microordenadores.

Al ser un compilador, el PASCAL es más rápido que el BASIC.

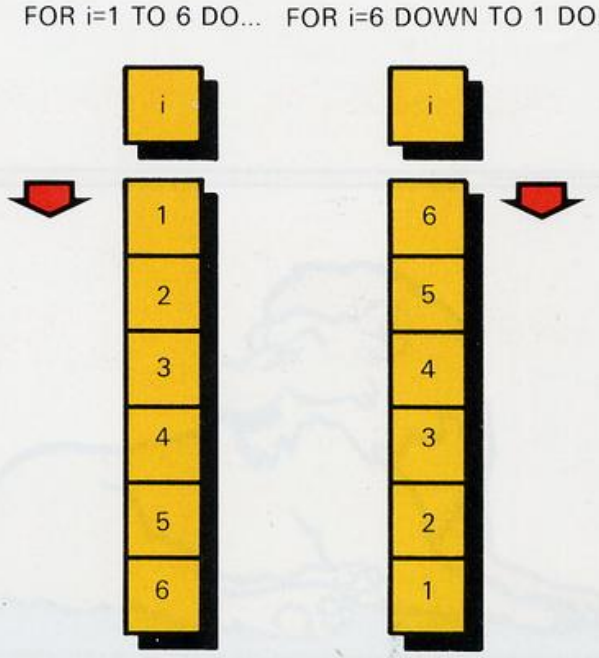


¡!

Antes de empezar a teclear un programa PASCAL hay que tener muy claras las ideas y un esquema de lo que pretendemos.

*

El PASCAL surgió para solucionar los problemas que presentaban los lenguajes de la primera época informática: dificultad de seguir un programa largo y de realizar correcciones posteriores.



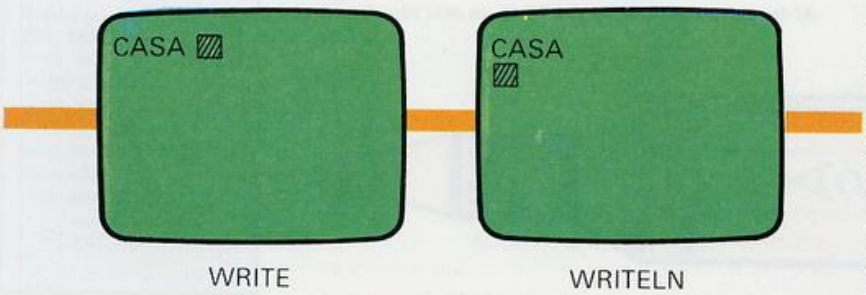
Con **FOR ... TO ... DO** *la variable de control del bucle va aumentando en una unidad, con* **FOR ... DOWNTO ... DO** *disminuye en una unidad.*

para que lo comprenda el usuario y no la máquina y esto se nota cuando trabajamos con él.

LOS PROGRAMAS

Quizá pueda parecer empezar la casa por el tejado el explicar primeramente la estructura de los programas sin conocer antes algunos comandos y su función. Sin embargo, por la estructura del

El comando **WRITE** *mantiene el cursor en la misma línea de impresión y* **WRITELN** *salta a la línea siguiente.*



PASCAL, al ser un compilador, es imposible introducir comandos directos, como ocurre con el BASIC. Todos los comandos deben ser introducidos formando parte de un programa y, por tanto, empleando su sintaxis y estructuración.

Todo programa PASCAL estará formado por las siguientes partes:

1. Asignación de un nombre al programa. Es lo primero a realizar y se utiliza el comando **PROGRAM** seguido del nombre.

10 PROGRAM sumayresta;

En el ejemplo se da el nombre "sumayresta" a un programa.

2. Especificación de los parámetros del programa. Estos pueden ser **INPUT** u **OUTPUT,** indicando que el programa realizará operaciones de entrada y salida, respectivamente. Se indica de la siguiente manera:

PROGRAM sumayresta (INPUT,OUTPUT);

Sin embargo, en el *HISOFT PASCAL* este tipo de indicación no sólo no es imprescindible sino que no es admitida, debido a las características propias de esta versión. Es necesario después de cada línea poner el «;».

3. Ahora llega la parte de declaración, en donde se enuncian primeramente el grupo de constantes (indicado con **CONST**) y el de variables (indicado con **VAR**) que serán utilizados a lo largo de todo el programa.

```
20 CONST
30     numa=3;
40     numb=2;
50 VAR
60     ressuma,resresta:INTEGER;
```

En esta ocasión se inicializan las constantes «base» y «suma» a «2» y «3» respectivamente. Por otro lado, se definen unas variables («exponente» y «resultado») que van a tomar valores enteros **(INTEGER),** las variables también pueden tomar valores reales **(REAL).**

Además de declarar constantes y variables en este apartado también son declarados otros parámetros que más adelante veremos.

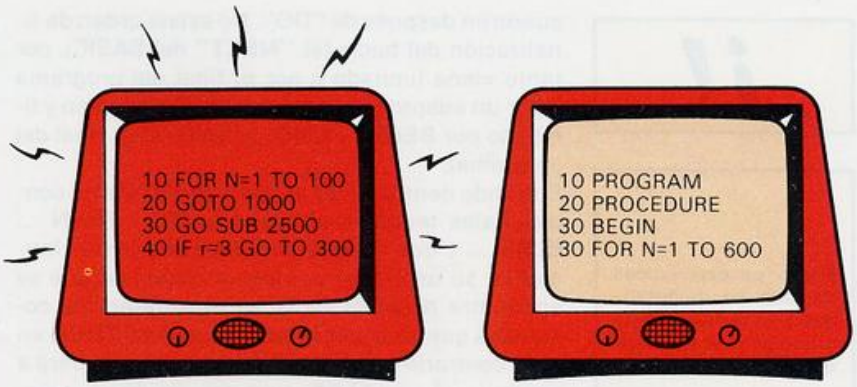
4. Aquí es donde entra el programa en sí. Se encabeza por la palabra **BEGIN** y se finaliza con **END.** Un punto («.») a continuación de esta palabra indica que es el final del texto del programa.

```
 70 BEGIN
 80     ressuma:=numa+numb;  halla la suma
 90     resresta:=numa-numb;  halla la resta
100     WRITE (ressuma,resresta);  imprime
resultado
110 END.
```

5. Sólo queda por mencionar los comentarios de ayuda al programador. No es un apartado en sí pero no puede ser introducido en ninguna de las partes anteriores. Se pueden colocar en cualquier punto del programa y, en la versión de HISOFT, debe estar encerrado entre corchetes.

Es muy común a la hora de programar en PAS-

CAL no situar todos los comandos sobre una misma columna, unos encima de otros, sino irlos introduciendo dentro de la línea según vamos avanzando en la programación. Así, en el ejemplo anterior **BEGIN** sobresale sobre lo escrito en las líneas inferiores. Esto es lo que se conoce por formato libre, y es aconsejable utilizarlo para una mejor comprensión de los programas.
Aunque en la versión de HISOFT PASCAL aparezcan los números de líneas, éstos sólo sirven para facilitar la edición y corrección de ellas no influyendo en la programación. De hecho, algunas versiones de otros ordenadores no incorporan los números.



Al no utilizar **GO TO**, *entre otros factores, un programa en PASCAL es más fácil de entender que uno en BASIC.*

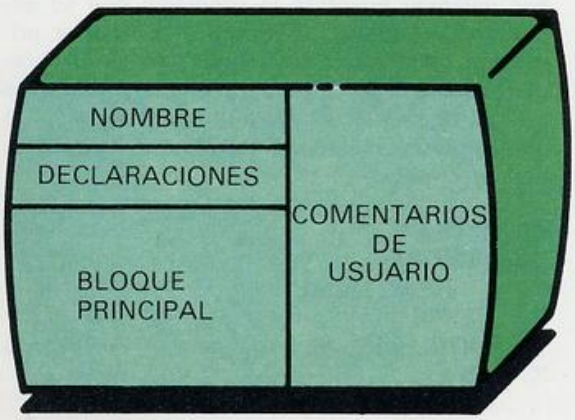
ESCRIBIMOS Y VEMOS

Ya en el ejemplo anterior vimos una sentencia de visualización por pantalla del resultado *(WRITE)*.
Si queremos una presentación mejor podemos hacer la siguiente modificación:
WRITELN (''SUMA='',ressuma);
WRITE (''RESTA='',resresta);
WRITELN escribe lo que tiene entre paréntesis pero al finalizar salta el cursor a la línea siguiente, al contrario de **WRITE** que deja el cursor a continuación del último literal representado.
Para la introducción de los datos disponemos del comando **READ** que permite la lectura de un dato por teclado y lo asigna a una variable.

READ (numa,numb)

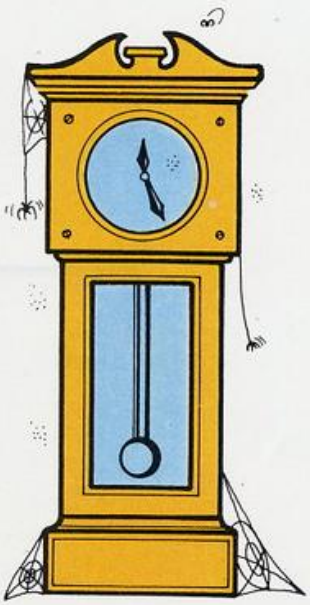
En este caso, el programa espera que introduzcamos dos datos separados por unos espacios. Es fácil observar que el primer parámetro se asignará a ''numa'' y el segundo a ''numb''.

En la programación del HISOFT-PASCAL hay que tener en cuenta los bloques que forman un programa.



También podemos provocar el salto de línea con **READLN** (lógico ¿verdad?), que como se podrá imaginar, después de la introducción del dato el cursor cambia a la línea siguiente.
Si queremos introducir las modificaciones presentadas en este apartado al programa anterior, deberemos tener en cuenta que hemos de eliminar la declaración de las constantes ''numa'' y ''numb'' (porque los valores se introducen ahora por teclado y no tendrán siempre el mismo valor) y cambiarlas a variables de tipo entero, situándolas a continuación de ''resresta''.

LOS BUCLES Y LAS SENTENCIAS DE CONTROL



El PASCAL dispone de un amplio repertorio de sentencias que puedan entrar dentro de este grupo. Vamos a conocerlas empezando por las más fáciles.
Como bucle propiamente dicho tenemos las sentencias **FOR...TO...DO** y **FOR...DOWNTO...DO** que son totalmente equivalentes al **''FOR...TO''** del BASIC. La primera de ellas incrementa el valor de una variable en una unidad hasta que se alcanza el número deseado. La segunda es prácticamente igual, pero provoca un decremento en una unidad. El PASCAL no admite saltos diferentes a la unidad.

FOR i:=1 TO 6 DO ...

En este caso, a la variable ''i'' se le asigna el valor inicial ''1'', y el valor máximo que tomará será ''6'', ejecutando todos los comandos que se en-

¡!

Como características destaca su modularidad y su posibilidad de manejar procedimientos.

*

Por sus características ocupa poca memoria, debido a lo cual puede ser incorporado en la mayoría de los microordenadores.

cuentren después de "DO". No existe orden de finalización del bucle (el **"NEXT"** del BASIC), por tanto viene limitado o por el final del programa o por un subprograma situado a continuación y limitado por **BEGIN** y **END;** (o **END.** si es final del programa).

Entrando dentro de las sentencias de control condicionales tenemos el conocido **IF ... THEN ... ELSE ...** , que no presenta ninguna particularidad en su uso: si se cumple la condición que se encuentra después de **IF** se ejecutarán los comandos que se encuentran después del **THEN** en caso contrario saltará y el programa continuará a continuación del **ELSE,** cuya colocación es opcional.

En el caso de menús es necesario incluir una sentencia **IF** por cada opción para en cada caso realizar la acción deseada. En PASCAL nos podemos ahorrar todos los condicionales por una sola instrucción: **CASE ... OF.** Así, si sabemos que una variable puede tomar diferentes valores (por ejemplo 6, 9 y 50) y dependiendo de cual sea hay que realizar unas acciones diferentes el problema se soluciona así:

```
CASE fondo OF
6:fondo:=0;
9:WRITE (fondo);
50:lado:=lado*5;
```

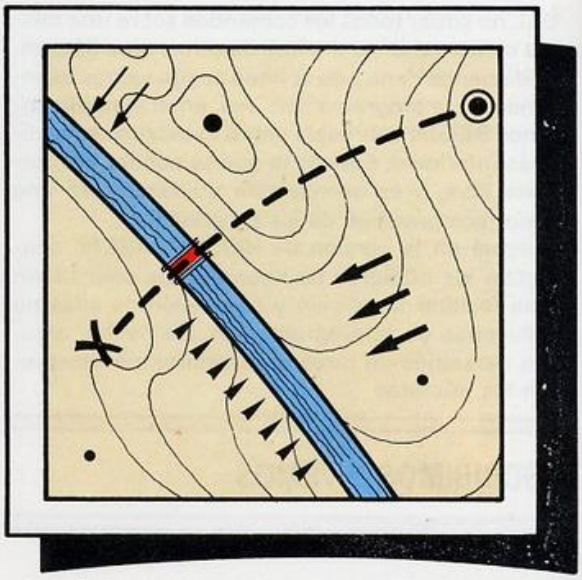
Examinamos el valor de la variable "fondo". Si vale "6" se le asigna a ella "0", si toma el valor "9" se imprime su valor y si es "50" a una variable que se utiliza en el programa ("dato") se le multiplica por "5" y se le asigna a ella misma. Una sentencia incorporada en muchos BASIC de los «micros» actuales pero que el Spectrum no «conoce» es **WHILE ... DO ...** Después de **WHILE** se inserta una condición y las sentencias que se encuentran después de **DO** se ejecutan continuamente mientras la condición sea cierta.

```
    ...
50  READ (num);
60  WHILE num<6 DO
70    BEGIN;
80      READ (num);
90    END;
    ...
```

En este fragmento del programa, primeramente se nos pide un número que le asignamos a "num", y mientras "num" vaya valiendo menos de "6" se ejecuta el subprograma a continuación de **DO** el cual nos vuelve a pedir otra introducción por teclado.

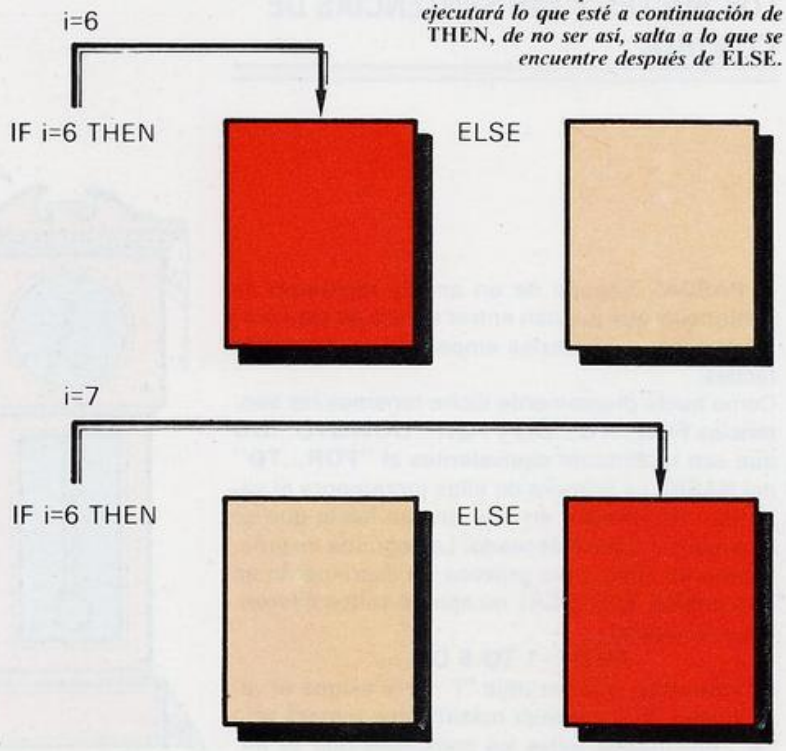
La última instrucción condicional que presenta el PASCAL es **REPEAT ... UNTIL,** en la cual se ejecutan las sentencias que se encuentra entre ambas palabras mientras la condición que se encuentra después de **UNTIL** sea falsa. Es decir, primero se ejecuta una vez y luego se realiza la pregunta sobre si continúa o no. Si ahora queremos que el fragmento del programa anterior haga lo mismo pero con estos comandos nuevos deberemos efectuar:

```
    ...
50  REPEAT;
60    BEGIN;
70      READ (num);
80    END;
90  UNTIL num>=6; { teclear ">" y "=", no
">="!!! }
    ...
```



Con el HISOFT-PASCAL podemos realizar dibujos al llevar incorporados los gráficos de tortuga, similares a los del LOGO.

Si se cumple la condición **IF** *se ejecutará lo que esté a continuación de* **THEN**, *de no ser así, salta a lo que se encuentre después de* **ELSE**.



Si "num" es menor que "6" el bucle se volverá a ejecutar, es decir, provoca el efecto contrario a **WHILE ... DO.**

Siempre que una variable va a ser utilizada solamente en una sentencia condicional es preciso declararla antes como **"BOOLEAN"** para indicar que va a ser empleada en una comparación. En el ejemplo anterior, si "num" no se utilizara en la introducción por teclado de un valor, y por tanto, se declarara como **"INTEGER"**, habría que efectuar:

VAR num:BOOLEAN;

El PASCAL también conoce la sentencia **GO TO** pero se aconseja utilizarla lo menos posible. El **GO TO** es un salto a un número, llamado etiqueta, que se declara previamente **(LABEL 100)**, nunca el salto será a un número de línea.

LOS PROCEDIMIENTOS

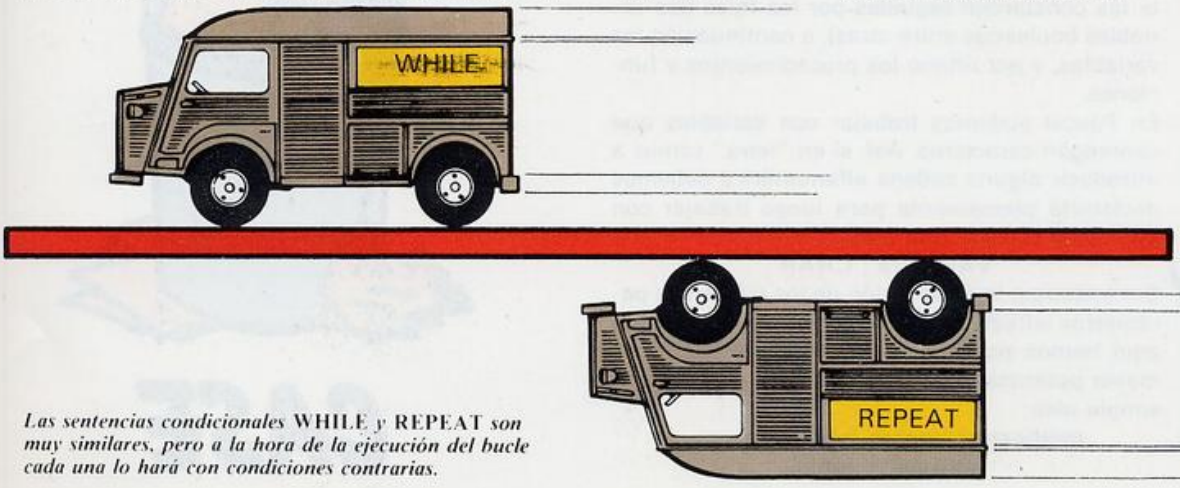
Un procedimiento es un fragmento de programa que realiza una sola función.

Los procedimientos y las funciones (una variante de aquellos) son equivalentes, aunque con mayor potencia, a las subrutinas del BASIC llamadas por **GOSUB**. Es la pieza fundamental en la realización de un programa en PASCAL.

Supongamos que tenemos entre manos un programa que realiza una serie de operaciones diferentes, lo ideal sería dividirlo en pequeños fragmentos que hicieran por separado una sola función. Cada uno de estos fragmentos es un procedimiento.

Vamos a hacer un programa que calculará el cuadrado de un número y para ello utilizaremos un procedimiento que será el encargado de realizar la operación. Para ello, después de realizados los preliminares que ya conocemos, debemos declarar el procedimiento y escribir los comandos para que realice nuestra función. Dentro del procedimiento debemos declarar todas las variables, constantes y parámetros que se utilicen, como si de un programa más se tratase.

```
10 PROGRAM cuadrado;
20 VAR numero:INTEGER;
30   PROCEDURE potencia; {Declaración procedimiento}
40     VAR resul:INTEGER; {Comienza el procedimiento}
50     BEGIN
60       resul:=numero*numero;
70       numero:=resul;
80     END;          {Fin del procedimiento}
90 BEGIN             {Comienzo del programa}
```



Las sentencias condicionales **WHILE** *y* **REPEAT** *son muy similares, pero a la hora de la ejecución del bucle cada una lo hará con condiciones contrarias.*

¡!

Es un lenguaje que ha sido pensado para que el usuario lo entienda fácilmente, y no para que lo entienda la máquina solamente.

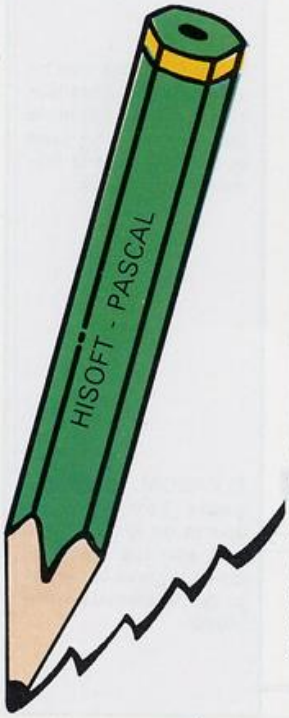
*

El PASCAL es un lenguaje compilado y, aparte de lo exigentes que son los compiladores, aporta la ventaja de su elevada velocidad.

A la hora de realizar las declaraciones debe seguirse siempre el mismo orden.

```
100   READ (numero);
110   potencia;        {Llamada al procedimiento}
120   WRITE (numero);
130 END.               {Fin del programa}
```

LAS DECLARACIONES



Aunque se ha estado mencionando a lo largo del capítulo, quizá convenga realizar un agrupamiento de todo lo mencionado.

Es necesario que las declaraciones sigan un orden para que todo marche bien; así, primero serán las etiquetas (para los **GO TO**) posteriormente las constantes seguidas por los tipos (las variables booleanas entre otras), a continuación las variables, y por último los procedimientos y funciones.

En Pascal podemos trabajar con variables que contengan caracteres. Así, si en "letra" vamos a introducir alguna cadena alfanumérica debemos declararla previamente para luego trabajar con ella como si fuera una más.

VAR letra : CHAR

Sin embargo, la declaración de los diferentes parámetros ofrece mucha más libertad de lo que aquí hemos podido ver, dando al PASCAL una mayor potencia en la programación. He aquí una simple idea:

medioanno=(enero,febrero,marzo,
abril,mayo,junio)

Podremos trabajar con la variable "medioanno" como si fuera casi un número. Observemos lo siguiente:

FOR mes=enero TO junio DO ...

Los valores de la variable "mes" variarán desde "enero" hasta "junio", ¡como si fuera un bucle con números!

LLEGO EL FIN

Esperamos que esta pequeña revisión haya sido suficiente, aunque tan solo sea para vislumbrar la potencia que puede aportar el PASCAL en cuanto se domine.

En las versiones estándar se hechan de menos los comandos gráficos (puesto que la finalidad inicial del lenguaje no era ésta y suponía un lujo para la época). Sin embargo, con el HISOFT tenemos esta posibilidad (además del sonido), que se hace casi obligatoria en los «micros» actuales, poniéndose así a nuestro alcance cualquier aspecto que queramos programar, sobre todo teniendo en cuenta que nos ofrece los gráficos de tortuga, que son totalmente similares a los del LOGO.

Un último consejo: el PASCAL es un lenguaje para hacerse un esquema antes de empezar a teclear en la máquina y tener las ideas muy claras. No así en el BASIC, donde directamente podemos empezar a teclear el programa.

La función CASE *nos supone un ahorro de sentencias* IF.

CASE

MOVIMIENTOS DE BLOQUES

En las próximas líneas discutiremos tres nuevos grupos de instrucciones denominados en las tablas del capítulo inicial de esta serie de intercambio, transferencia y búsqueda de bloques, a los cuales añadiremos antes de finalizar, el grupo lógico de 8 bits. Comencemos sin más, por el primero de ellos.

EL GRUPO DE INTERCAMBIO

Las instrucciones que lo componen provocan el intercambio de la información contenida en los registros implicados. No tiene sentido hablar de origen y destino, pues el intercambio es biyectivo. Consideremos la primera de ellas: **EX DE,HL** (el mnemónico EX responde a la abreviatura de la palabra inglesa *EXchange,* canje o intercambio). Mediante ella, el valor almacenado en el par DE se transfiere al par HL, y simultáneamente, el contenido de HL pasa a DE.

LOS REGISTROS ALTERNATIVOS

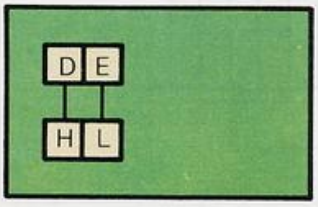
Las instrucciones de intercambio habilitan el uso del juego de registros alternativos, el cual se utiliza habitualmente en C/M para conservar valores o direcciones de interés que requieran una fácil y rápida recuperación, a la par de seguridad ante una posible corrupción como puede ocurrir, por ejemplo, en el stack.

EX AF,A'F', efectúa el canje de los valores contenidos en los pares AF y A'F'. Finalmente, completa el grupo de intercambio una potente ins-

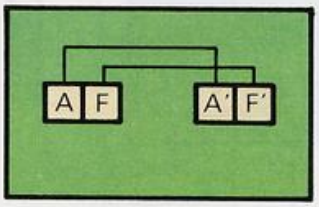
Siguiendo el mismo mecanismo, tres instrucciones se encargan de intercambiar información con el stack: **EX(SP),HL EX(SP),IX EX(SP),IY,** de manera tal, que permiten canjear el valor actual de los registros HL, IX o IY con la última entrada efectuada en la pila. Y lo que es más importante: el contenido del Stack Pointer SP no resulta alterado.

ENSAMBLADOR	CODIGO MAQUINA	Nº BYTES	CICLOS RELOJ	CICLOS MAQUINA
EX DE,HL	1 1 1 0 1 0 1 1	1	4	1
EX AF,AF'	0 0 0 0 1 0 0 0	1	4	1

INDICADORES: NO AFECTADOS



Z 80



Z 80

¡!

Al ejecutar la instrucción OR s, se efectúa la suma lógica entre los bits correspondientes de los dos octetos implicados.

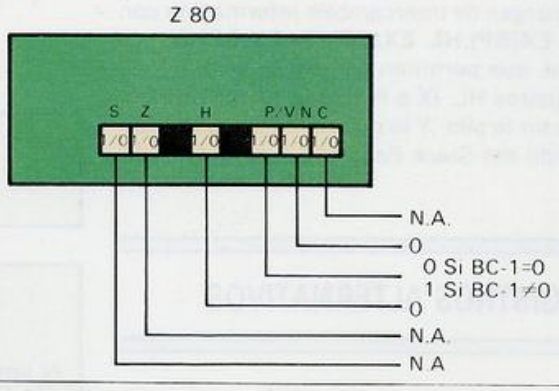
*

Las instrucciones **LDI, LDD, CPI, CPD** carecen de autorrepetición.

*

Si deseamos examinar un área determinada de memoria para saber si un valor particular está presente en ella, situaremos el origen en el par HI, el número de bits a comparar en BC, y en el acumulador A el valor a ser contrastado.

ENSAMBLADOR	CODIGO MAQUINA	Nº BYTES	CICLOS RELOJ	CICLOS MAQUINA
LDI	11101101 10100000	2	16	4
LDD	11101101 10101000	2	16	4

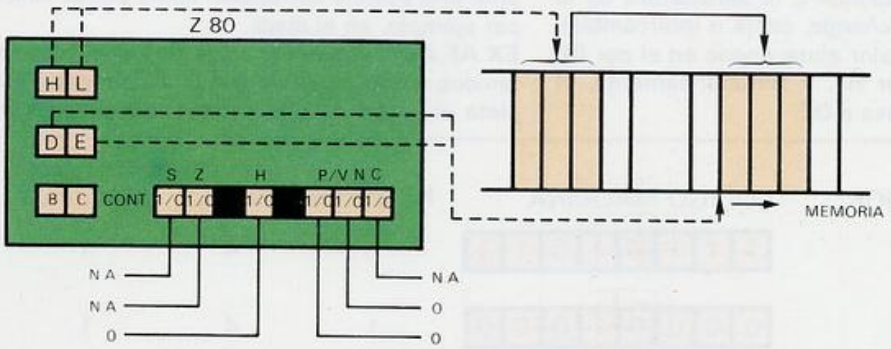


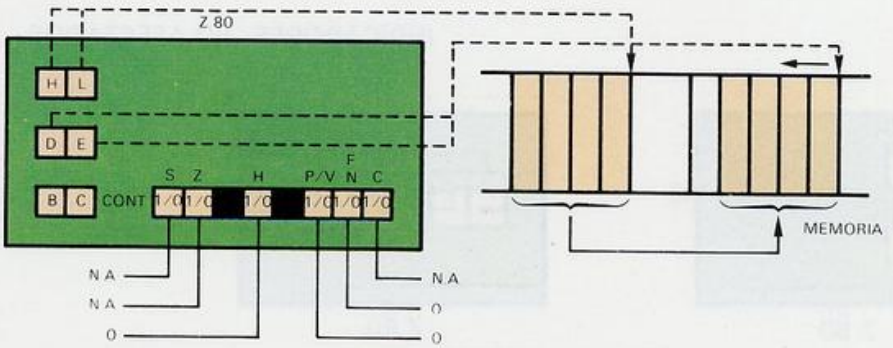
trucción: **EXX**, la cual de una sola vez intercambia los datos almacenados en los registros BC, DE y HL con los contenidos de B'C', D'E' y H'L', respectivamente.

MANEJANDO BLOQUES

Entre las instrucciones desarrolladas para el microprocesador Z80, ocho fueron las previstas para gestionar la transferencia y búsqueda de bloques. Las primeras permiten al programador trasladar un área de memoria de una zona a otra de ésta. Las segundas examinan si una determinada configuración se haya presente dentro de un bloque de datos.

ENSAMBLADOR	CODIGO MAQUINA	Nº BYTES	CICLOS RELOJ	CICLOS MAQUINA
LDIR	11101101 10110000	2	21 BC≠0 16 BC=0	5 4
LDDR	11101101 10111000	2	21 BC≠0 16 BC=0	5 4





¡!

Las tres instrucciones que se encargan de intercambiar información con el stack son: **EX(SP),HL, EX(SP),IX, EX(SP),IY.**

*

Entre las instrucciones desarrolladas para el microprocesador Z80, ocho fueron previstas para gestionar la transferencia y búsqueda de bloques.

Cuando efectuemos la transferencia de un bloque, la dirección de origen de éste debemos cargarla previamente en el par HL, y la dirección de destino, en el par DE. Por su parte, los registros BC han de contener el número de bytes a ser transferidos.

Si nuestro objetivo es examinar un área determinada de memoria al objeto de confirmar o no, si un valor particular está presente en ella colocaremos el origen de ésta en el par HL, el número de bytes a comparar en BC, y en el acumulador A, el valor particular a ser contrastado.

Podemos clasificarlas en dos subgrupos:

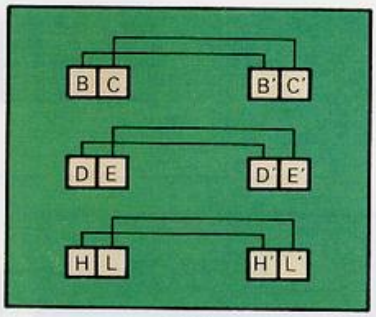
1) Funcionamiento automático **(LDIR, LDDR, CPIR, CPDR).**

Estas instrucciones ejecutan cíclicamente su trabajo hasta que una determinada condición les indica que ha finalizado éste. Por ejemplo, supongamos que deseamos transferir un bloque de una zona a otra mediante la orden LDIR (LD-cargar, I-incrementar, R-repetir). Internamente, el microprocesador efectúa las siguientes operaciones:

— T1: Carga el contenido de la posición de memoria direccionada por DE (como sabemos se tra-

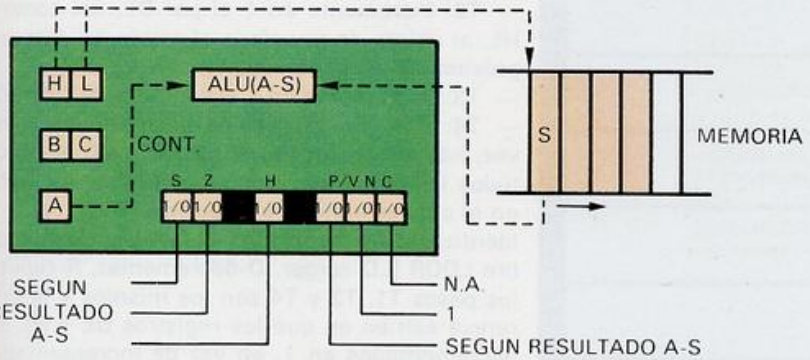
ENSAMBLADOR	CODIGO MAQUINA	Nº BYTES	CILOS RELOJ	CICLOS MAQUINA
EXX	11011001	1	4	1

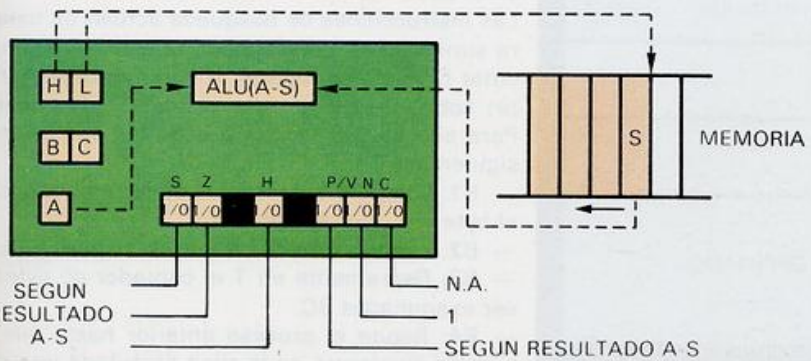
INDICADORES: NO AFECTADOS



Z 80

ENSAMBLADOR	CODIGO MAQUINA	Nº BYTES	CICLOS RELOJ	CICLOS MAQUINA
CPIR	11101101 10110001	2	21 BC≠0 y A≠(HL) 16 BC=0 ó A=(HL)	5 4
CPDR	11101101 10111001	2	21 BC≠0 y A≠(HL) 16 BC=0 ó A=(HL)	5 4





¡!

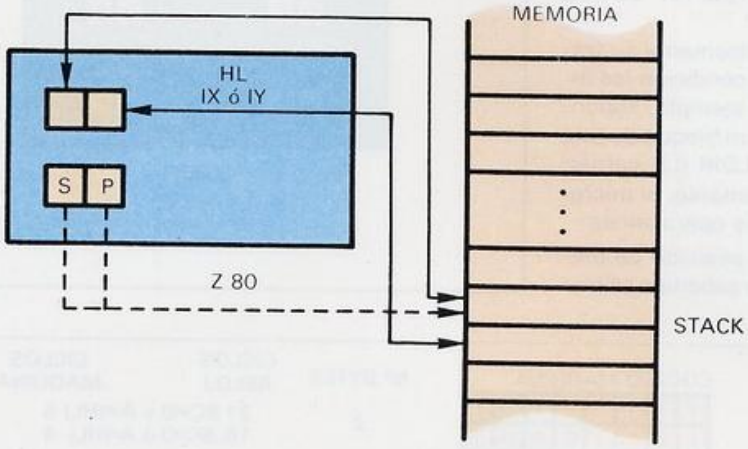
Las instrucciones que componen el grupo de intercambio provocan el intercambio de la información contenida en los registros.

*

Las instrucciones **LDIR, LDDR, CPIR, CPDR** ejecutan cíclicamente su trabajo hasta que una determinada condición les indica que ha finalizado éste.

ENSAMBLADOR	CODIGO MAQUINA	Nº BYTES	CICLOS RELOJ	CICLOS MAQUINA
EX (SP),HL	1 1 1 0 0 0 1 1	1	19	5
EX (SP),IX	1 1 0 1 1 1 0 1 1 1 1 0 0 0 1 1	2	23	6
EX (SP),IY	1 1 1 1 1 1 0 1 1 1 1 0 0 0 1 1	2	23	6

INDICADORES: NO AFECTADOS



OR s				
OPERANDO	HEX.	DEC.	INDICADOR	
n	F6,n	246,n	S	1 si resultado – 0 si resultado ++
A	B7	183	H	1 si resultado 0 0 en caso contrario
B	B0	176		
C	B1	177	H	0
D	B2	178	P/V	1 paridad par 0 paridad impar
E	B3	179		
H	B4	180	N	0
L	B5	181	C	0
(HL)	B6	182		
(IX+d)	DD,B6,d	221,182,d	S OPERANDO	
(IY+d)	FD,B6,d	253,182,d		

ta de la dirección destino) con el octeto señalado por HL (dirección origen o fuente).

— T2: Incrementa en 1 el par DE, así como el HL, al objeto de transferir el siguiente dato a la próxima posición de destino.

— T3: Decrementa en 1 el contador de bytes BC.

— T4: El mismo proceso es ejecutado una y otra vez, hasta que por fin, el par BC es 0, es decir, todos los datos han sido transferidos, momento en el cual termina la instrucción.

Idénticas consideraciones es factible efectuar sobre LDDR (LD-cargar, D-decrementar, R-repetir): los pasos T1, T3 y T4 son los mismos y la diferencia estriba en que los registros DE y HL son decrementados en 1, en vez de incrementados, en el paso T2.

Las instrucciones de búsqueda actúan de manera similar. Sea, por ejemplo, nuestro objeto ejecutar CPIR (CP-comparar, I-incrementar, R-repetir) sobre un bloque de memoria determinado. Para ello el Z80 llevará a cabo las operaciones siguientes:

— B1: Compara el contenido del acumulador con el byte direccionado por HL.

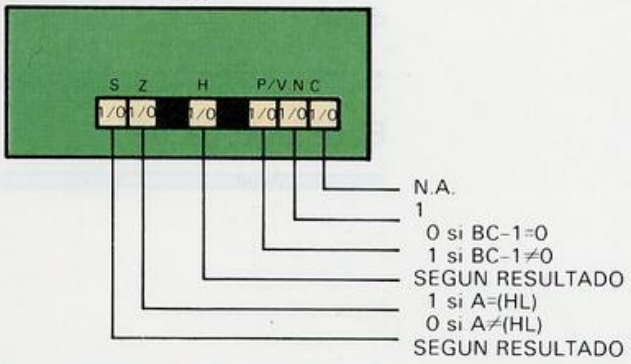
— B2: Incrementa en 1 el par de registros HL.

— B3: Decrementa en 1 el contador de bytes a ser examinados BC.

— B4: Repite el proceso anterior hasta que, o bien se encuentra en la zona escrutada un valor

idéntico al del acumulador, o bien, no quedan más bytes a ser comparados, circunstancia satisfecha cuando BC es 0.
CPDR (CP-comparar, D-decrementar, R-repetir) es a las instrucciones de búsqueda lo que LDDR es a las de transferencia de bloques. La única diferencia en cuanto a funcionamiento con CPIR reside en el paso B2, donde el par HL es decrementado en vez de incrementado en 1.
2) Mecanismo no automático **(LDI, LDD, CPI, CPD).**
Estas cuatro instrucciones tienen en común con las del grupo anterior los tres primeros pasos indicados, pero carecen de la autorrepetición. Por tanto, al construir un programa que las implemente deberemos hacer uso de ellas, una vez por cada byte a ser transferido o comparado.

ENSAMBLADOR	CODIGO MAQUINA	Nº BYTES	CICLOS RELOJ	CICLOS MAQUINA
CPI	11101101 10100001	2	16	4
CPD	11101101 10101001	2	16	4



LAS INSTRUCCIONES LOGICAS

Llegado este apartado os recomendamos antes de proseguir con él repasar el capítulo dedicado al Algebra de Boole, punto de partida de todo lo que seguirá a continuación.
Las instrucciones lógicas enfrentan dos octetos entre sí efectuando la operación (AND, OR, XOR) entre sus bits individuales tomándolos por parejas formadas por aquellos que ocupan el mismo lugar dentro de los bytes considerados.
El primero de ellos ha de estar contenido en el acumulador A, y el segundo en un registro o localización de memoria específica. El resultado siempre se recoge en A.

EL OPERADOR AND

En la figura está representada su tabla de verdad. Como observaremos, efectúa el producto lógico entre los bits implicados, es decir, el resultado es 1 si, y sólo si, los dos son 1, siendo 0 si al menos uno de ellos es 0.
Una de las operaciones más habituales en los programas en C/M consiste en aplicarlo sobre el mismo acumulador, es decir, implementar la ins-

AND s

OPERANDO	HEX.	DEC.	INDICADOR	
n	E6,n	230,n	S	1 si resultado - 0 si resultado +
A	A7	167	Z	1 si resultado 0 0 en caso contrario
B	A0	160	H	1
C	A1	161	P/V	1 paridad par 0 paridad impar
D	A2	162	N	0
E	A3	163	C	0
H	A4	164		
L	A5	165		
(HL)	A6	166		
(IX+d)	DD,A6,d	222,166,d	S OPERANDO	
(IY+d)	FD,A6,d	253,166,d		

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

XOR	0	1
0	0	1
1	1	0

trucción AND A. Con ello, el contenido de este registro se mantiene inalterado, pero modifica algunos bits del registro F, de tal manera, que es posible identificar si el valor almacenado en A es 0, negativo o tiene un número par de unos (paridad par).

EL OPERADOR OR

Al ejecutar la instrucción OR s, donde s es cualquier valor almacenado en los registros o posiciones de memoria que se detallan en las figuras, se efectúa la suma lógica entre los bits correspondientes de los dos octetos implicados.
Hemos de tener presente que en toda operación lógica no se producen acarreos, es decir, el resultado obtenido tras aplicar cualquiera de los operadores sobre una pareja de bits, no tiene ninguna influencia sobre los siguientes, siendo tratadas de forma independiente.
Al igual que antes analizando el contenido del registro F tras implementar la instrucción OR A, estamos en condiciones de precisar si el dato almacenado en el acumulador es negativo, cero o tiene paridad par.

OR EXCLUSIVO XOR

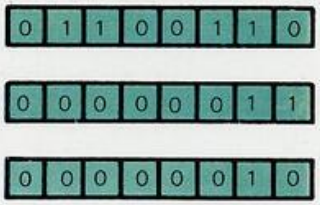
El resultado de este operador aplicado sobre dos octetos determinados conduce a la comparación lógica de ambos, es decir, cuando enfrentamos dos bits se obtiene 1 si son diferentes, y 0 en el caso de ser iguales.
Muchos programas utilizan la instrucción XOR A como sustituto de LD A,0, siempre y cuando no tenga importancia modificar el contenido del registro F, puesto que se consigue ahorrar un byte por dicho sistema.

XOR s				
OPERANDO	HEX.	DEC.	INDICADOR	
n	EE,n	238,n	S	1 si resultado – 0 si resultado +
A	AF	175	Z	1 si resultado 0 0 en caso contrario
B	A8	168		
C	A9	169	H	0
D	AA	170	P/V	1 si paridad par 0 si paridad impar
E	AB	171		
H	AC	172	N	0
L	AD	173	C	0
(HL)	AE	174		
(IX+d)	DD,AE,d	221,174,d	S OPERANDO	
(IY+d)	FD,AE,d	253,174,d		

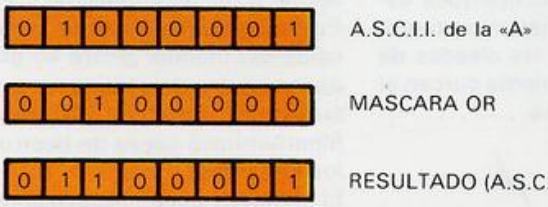
MASCARAS

Una de las utilidades más importantes ligada al empleo de las instrucciones lógicas es la de permitir una técnica de programación basada en las máscaras (en inglés *mask*). En la tabla podemos encontrar algunos ejemplos relacionados con ellas.

1. Eliminar ciertos bits dentro de un byte (Ej. In seis superiores)



2. Paso de minúscula a mayúscula



Quizá fuera más preciso utilizar el término «filtro», pues el objetivo final de dicha técnica consiste en seleccionar ciertos bits de interés para nosotros dentro de un octeto, obviando o inhibiendo los restantes.

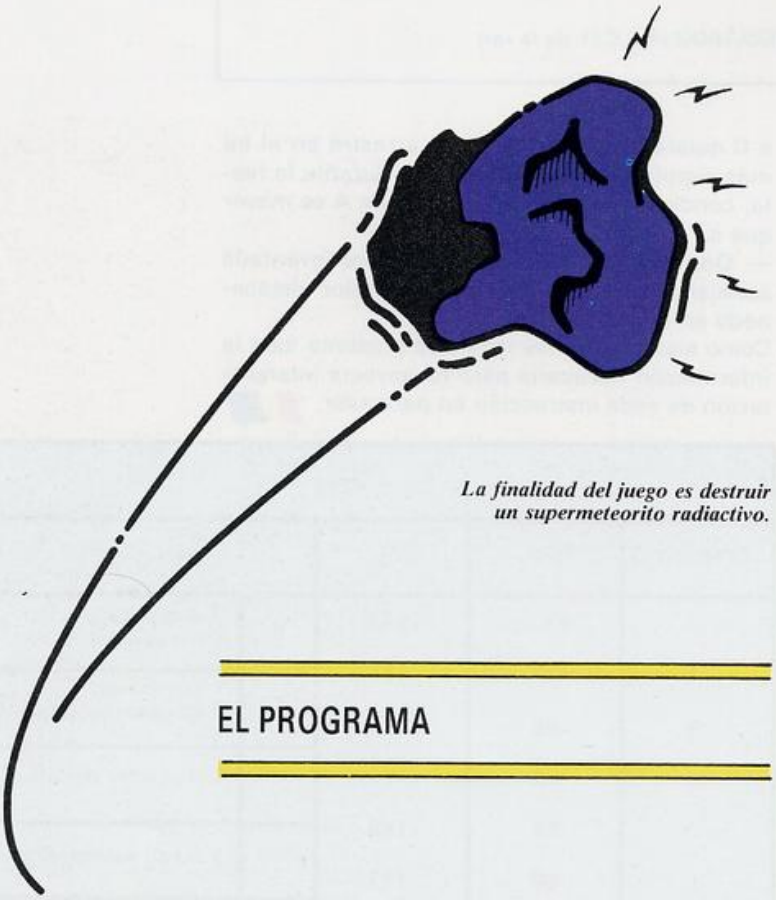
COMPARACIONES

Para completar el grupo lógico de 8 bits nos queda analizar las instrucciones de código mnemónico CP (*ComPare,* comparar), las cuales confrontan el contenido del acumulador con un byte s determinado por el valor almacenado en un registro o posición de memoria específica.
Como sabemos, el resultado de una comparación solamente ofrece dos posibilidades: o las dos cantidades son iguales o son diferentes. Cuando se trata del segundo caso puede interesarnos conocer cual es la mayor (o cual es la menor).
El Z80 soluciona la papeleta con habilidad ejecutando internamente la resta A–s y activando algunos indicadores los cuales delatan sin duda el resultado de la comparación, manteniendo inalterados los contenidos tanto del acumulador como del byte s. Comprobémoslo:
— Si la bandera de cero Z está alzada, es decir, contiene un 1, entonces la resta A–s fue 0, y por tanto, A y s son iguales.
— En caso contrario Z contendrá un 0. Nos fijamos ahora en el indicador de acarreo C. Si está a 0 quiere decir que no hubo arrastre en el bit más significativo del acumulador durante la resta, condición que se cumple cuando A es mayor que s.
— De no ser así, la bandera C estará levantada señalando que s es mayor que el valor almacenado en el acumulador.
Como siempre, en las tablas se presenta toda la información necesaria para la correcta interpretación de cada instrucción en particular.

CP s				
OPERANDO	HEX.	DEC.	INDICADORES	
n	FE,n	254,n	S	1 si resultado – 0 si resultado +
A	BF	191	Z	1 si resultado 0 0 en caso contrario
B	B8	184		
C	B9	185	H	1 si acarreo del BIT 3
D	BA	186	P/V	1 si hay sobrepasamiento
E	BB	187		
H	BC	188	N	1
L	BD	189	C	1 si hay acarreo
(HL)	BE	190		
(IX+d)	DD,BE,d	221,190,d	S OPERANDO	
(IY+d)	FD,BE,d	223,190,d		

RADAR

Nuestro monitor o televisor acaba de convertirse en una pantalla de radar de seguimiento. Podemos vigilar todos los movimientos de cualquier estrella del universo y, lo que es más importante: salvaguardar nuestro planeta de las oleadas de meteoritos radiactivos que últimamente surcan el espacio exterior cercano a la Tierra.



La finalidad del juego es destruir un supermeteorito radiactivo.

EL PROGRAMA

Una vez el programa ha sido debidamente introducido y ejecutado, aparecerá en la pantalla un menú de selección de dificultad. Dicho menú sólo posee dos opciones: fácil o difícil. La complejidad del juego dependerá del número de estrellas estáticas que surjan en el monitor. Lógicamente, si el nivel de juego es el complicado, aparecerán más estrellas que en el nivel de menos dificultad. La finalidad del juego consiste en destruir un super meteorito radiactivo que aparecerá en pantalla inmediatamente después de que hayan sido posicionadas las estrellas estáticas. Para ello, nuestro monitor radar está provisto de un indicador de posición a modo de cursor.

Pulsando cualquier tecla (excepto el 0), el indicador del monitor girará 45 grados en el sentido de las agujas del reloj, a medida que prosigue en su avance. Pulsando «0», dispararemos un rayo hiperlumínico capaz de destruir al más terco de los meteoros.

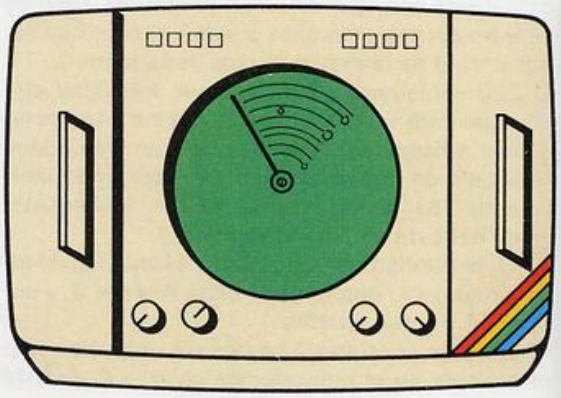
El indicador puede desplazarse en cualquier sentido del plano dependiendo siempre de su rotación de 45 grados. Si en nuestro avance, colisionamos con una estrella estática, el indicador variará de rumbo 45 grados, desviándose de la ruta de interceptación, aunque no se nos producirá daño alguno.

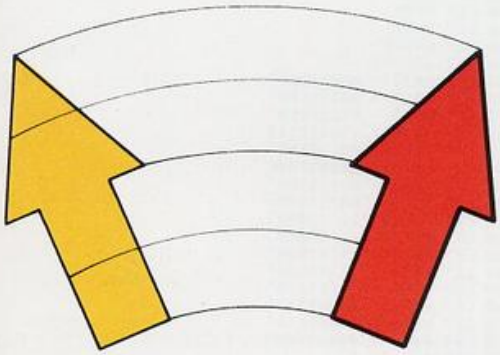
Para llevar a buen término la misión disponemos solamente de 100 segundos. Transcurrido el cual podemos considerarnos destruidos.

INTRODUCCION DEL PROGRAMA

La introducción del programa será incluso algo más sencilla de lo habitual, puesto que no exis-

Nuestro monitor o televisor acaba de convertirse en una pantalla de radar.





ten caracteres gráficos numéricos. Sólo tendremos que preocuparnos de sustituir las letras subrayadas en las líneas 1500 y 1700 del listado, por los caracteres gráficos correspondientes.
En lo referente a la grabación del programa, podemos llevarla a cabo mediante el comando **SAVE "RADAR"**, o bien **SAVE "RADAR" LINE 10** si deseamos que el programa se autoejecute al finalizar su carga.

***Para el giro de 45 grados del cursor emplearemos cualquier tecla menos* 0.**

Para la grabación del programa podemos utilizar el comando **SAVE "RADAR"**, o bien **SAVE "RADAR" LINE 10** si se desea la autoejecución del programa cada vez que se cargue.

Las letras que aparecen subrayadas en el listado corresponden a los gráficos de las teclas afectadas.

Aunque las teclas empleadas para el manejo del programa son todas las del teclado, este es enormemente sencillo de controlar.

```
 5 REM J.M. MAYORAL SERRANO
 10 POKE 23658,8
 20 GO SUB 790
 30 GO SUB 1000
 40 GO SUB 940
 50 IF INT T>1000 THEN  GO TO 1800
 60 GO SUB 110
 70 GO SUB 540
 80 IF H=1 THEN  GO TO 110
 90 GO SUB 650
100 GO TO 40
110 LET A$=INKEY$
120 IF A$="" THEN  RETURN
130 IF A$="0" THEN  GO TO 170
140 LET K=K+1
150 IF K>8 THEN  LET K=1
160 RETURN
170 LET XL=X*8+4
180 LET YL=175-Y*8-4
190 GO SUB 470
200 PLOT XL,YL
210 LET DX=0
220 IF V(K)=1 THEN  LET DX=255-XL
230 IF V(K)=-1 THEN  LET DX=-XL
240 LET DY=0
250 IF W(K)=1 THEN  LET DY=-YL
260 IF W(K)=-1 THEN  LET DY=175-YL
270 IF V(K)*W(K)=0 THEN  GO TO 300
280  IF ABS DX<ABS DY THEN  LET DY=ABS DX*SGN DY: GO
TO 300
290 LET DX=ABS DY*SGN DX
300 DRAW DX,DY
310 PLOT XL,YL
320 BEEP .01,40
330 DRAW  OVER 1;DX,DY
340 IF H=0 THEN  RETURN
350 LET MX=B*8+4: LET MY=175-A*8-4
360 FOR I=1 TO RND*5+20
370 PLOT MX,MY
380 LET DX=10-RND*20
390 IF MX+DX>255 OR MX+DX<0 THEN  GO TO 450
400 LET DY=10-RND*20
410 IF MY+DY>175 OR MY+DY<0 THEN  GO TO 450
420 PLOT MX,MY
430 DRAW DX,DY
440 BEEP .01,10
450 NEXT I
460  GO TO 1810
470 LET H=0
480 LET DY=A-Y
490 LET DX=B-X
500 IF W(K)*DX<>V(K)*DY THEN  RETURN
510 IF ABS V(K)*SGN DX<>V(K) OR ABS W(K)*SGN DY<>W(K
) THEN  RETURN
520 LET H=1
530 RETURN
540 IF NB1=0 THEN  PRINT AT Y,X;" "
550 LET X=X+V(K)
560 LET Y=Y+W(K)
570 IF X<0 THEN  LET X=31
580 IF X>31 THEN  LET X=0
590 IF Y<0 THEN  LET Y=21
600 IF Y>21 THEN  LET Y=0
610 IF ATTR (Y,X)<>15 THEN  BEEP .1,0: LET NB1=1: GO
TO 140
620 PRINT AT Y,X;M$(K)
630 LET NB1=0
640 RETURN
650 IF RND>1.05-DF/20 THEN  LET Z=Z+1
660 IF Z>8 THEN  LET Z=1
670 IF NB2=0 THEN  PRINT AT A,B;" "
680 LET A=A+V(Z)
690 LET B=B+W(Z)
700 IF B<0 THEN  LET B=31
710 IF B>31 THEN  LET B=0
720 IF A<0 THEN  LET A=21
730 IF A>21 THEN  LET A=0
740  IF ATTR (A,B)<>15 THEN  BEEP .1,-10: LET NB2=1:
LET Z=Z+1: GO TO 660
750 PRINT AT A,B;W$(R+1)
760 LET R=NOT R
770 LET NB2=0
780 RETURN
790 PAPER 1
800 BORDER 1
810 INK 7
820 CLS
830 PRINT AT 15,1;"INTRODUCE NIVEL DE DIFICULTAD"
840  PRINT
850 PRINT TAB 3;"1 (Facil) ///  2 (Dificil)"
860  LET k$=INKEY$
870 IF k$="" THEN  GO TO 860
880  IF CODE k$<49 OR CODE k$>50 THEN  GO TO 860
890 BEEP .1,40
900 LET df=VAL k$
910  IF df=2 THEN  LET df=10
920 CLS
930 RETURN
940 LET T=(PEEK 23672+256*PEEK 23673+65536*PEEK 2367
4)/5
950 PRINT AT 0,0;INT T/10;"  "
960 RETURN
970 IF RND>.1+DF/50 THEN  RETURN
980 PRINT AT RND*21,RND*31; INK 6;"*"
990 RETURN
1000 DIM W(8)
1010 DIM V(8)
1020 DIM S$(8,8)
1030 DIM R$(8,8)
1040 DATA 0,1,1,1,1,0,1,-1,0,-1,-1,-1,-1,0,-1,1
1050 FOR I=1 TO 8
1060 READ W(I),V(I)
1070 NEXT I
1080 LET K=1
1090 LET X=20
1100 LET Y=10
1110 LET V=V(K)
1120 LET W=W(K)
1130 LET S$(1)="00011000"
1140 LET S$(2)="00111100"
1150 LET S$(3)="01111110"
1160 LET S$(4)="11111111"
1170 LET S$(5)="00111100"
1180 LET S$(6)="00111100"
1190 LET S$(7)="00111100"
1200 LET S$(8)="00111100"
1210 LET R$(1)="11111000"
1220 LET R$(2)="11110000"
1230 LET R$(3)="11111000"
1240 LET R$(4)="11111100"
1250 LET R$(5)="10111110"
1260 LET R$(6)="00011111"
1270 LET R$(7)="00001110"
1280 LET R$(8)="00000100"
1290 FOR I=1 TO 8
1300 LET A=0: LET B=0: LET C=0
1310 LET D=0: LET E=0: LET F=0
1320 FOR J=1 TO 8
1330 LET A=A*2+VAL (S$(I,J))
1340 LET B=B*2+VAL (S$(9-J,I))
1350 LET C=C*2+VAL (S$(J,I))
1360 LET D=D*2+VAL (R$(I,J))
1370 LET E=E*2+VAL (R$(9-J,I))
1380 LET F=F*2+VAL (R$(9-I,9-J))
1390 GO SUB 970
1400 NEXT J
1410 POKE USR "A"+I-1,A
1420 POKE USR "B"+8-I,A
1430 POKE USR "C"+I-1,B
1440 POKE USR "D"+I-1,C
1450 POKE USR "E"+I-1,D
1460 POKE USR "F"+8-I,D
1470 POKE USR "G"+I-1,E
1480 POKE USR "H"+I-1,F
1490 NEXT I
1500 LET M$="CHBFDEAG"
1510 LET A=10
1520 LET B=10
1530 LET Z=INT (RND*8)+1
1540  POKE USR "I"+0,BIN 00100000
1550 POKE USR "I"+1,BIN 01000010
1560 POKE USR "I"+2,BIN 00100101
1570 POKE USR "I"+3,BIN 00011000
1580 POKE USR "I"+4,BIN 00011000
1590 POKE USR "I"+5,BIN 10100100
1600 POKE USR "I"+6,BIN 01000010
1610 POKE USR "I"+7,BIN 00000100
1620 POKE USR "J"+0,BIN 00001110
1630 POKE USR "J"+1,BIN 10001000
1640 POKE USR "J"+2,BIN 10001000
1650 POKE USR "J"+3,BIN 11111000
1660 POKE USR "J"+4,BIN 00011111
1670 POKE USR "J"+5,BIN 00001001
1680 POKE USR "J"+6,BIN 00001001
1690 POKE USR "J"+7,BIN 00111000
1700 LET W$="IJ"
1710 LET R=0
1720 INK 7
1730 LET NB1=0
1740 LET NB2=0
1750 POKE 23674,0
1760 POKE 23673,0
1770 POKE 23672,0
1780 LET H=0
1790 RETURN
1800 IF H<>1 THEN  GO SUB 1840
1810 PRINT AT 13,9;" BUEN DISPARO "
1820 PRINT
1830 GO TO 1860
1840 PRINT AT 15,5; PAPER 6; INK 9;" EL TIEMPO HA ACA
BADO "
1850 PRINT
1860 PRINT TAB 5;" OTRA PARTIDITA? S/N"
1870 PRINT
1880 PRINT TAB 10;"TIEMPO= ";INT T/10
1890  LET K$=INKEY$
1900 IF K$="" THEN  BEEP .1,40: GO TO 1890
1910 IF K$="S" THEN  RUN
1920 IF K$="N" THEN  GO TO 10000
1930 BEEP .1,45
1940 GO TO 1890
```