MSX

CODIGO MAQUINA
PROGRAMACION PRACTICA

STEVE WEBB

ra-ma

MSX
CODIGO MAQUINA
PROGRAMACION PRACTICA

STEVE WEBB

CODIGO MAQUINA
PROGRAMACION PRACTICA

STEVE WEBB

Título de la obra original

PRACTICAL MSX
MACHINE CODE
PROGRAMMING

Publicado en Gran Bretaña en 1985
por Virgin Books Ltd, 61-63 Portobello
Road, London W11 3DD.

ISBN 0863690742

Edición en Español

MSX CODIGO MAQUINA
PROGRAMACION PRACTICA

ISBN 84-86381-03-7

Editado por RAMA
Chiquinquira, 28 (COCUY)
28033 MADRID
ESPAÑA

Consultas referentes al libro
RAMA, Chiquinquira, 28. Teléfono: 764.50.95 - 28033 Madrid

Traducción y Composición: CONORG, S.A.

Signo Impresores, S. A. - Albasanz, 27 - Madrid.
Depósito legal: M. 13163 - 1985.
I. S. B. N.: 84-86381-03-7

Contenidos

Introducción 9

¿Qué es Código Máquina? 11

Equivalentes en Código Máquina
de las Instrucciones en BASIC 17

Almacenando los "Códigos de Operaciones"
en Memoria 31

Entendiendo la Imagen en Pantalla 41

Figuras y Motivos Visivos 'Sprites' 49

Programa Invasor del Espacio 55

Facilidades Avanzadas del MSX 75

Algunas Rutinas Aprovechables 89

Apéndices 99

Me gustaría aprovechar esta oportunidad para agradecer a Cat que ha empleado una gran cantidad de tiempo y esfuerzo en editar este libro. Mis gracias también se hacen extensivas a Sue Davies y a ZOE que hizo cada día más feliz que el anterior.

Introducción

La intención de este libro es introducirte en la programación en el Código Máquina de los ordenadores MSX. Los capítulos han sido escritos en un orden lógico y es importante que leas y entiendas cada uno de ellos antes de pasar al siguiente.

La programación en Código Máquina no es tan compleja como probablemente te hayan inducido a creer. Si tienes un conocimiento profundo de BASIC, serás capaz de captar muy rápidamente los conceptos sobre programación en Código Máquina.

Los primeros capítulos con la teoría del Código Máquina, te muestran qué números están almacenados; responden a la pregunta ¿qué es Código Máquina?; describen las equivalencias en Código Máquina de la mayoría de las instrucciones BASIC y te enseñan la organización del ordenador MSX.

El capítulo principal te explica cómo escribir un juego muy simple en Código Máquina. Te muestra cómo un programa puede primero describirse mediante un diagrama de flujo, y luego transcribir los bloques de ese diagrama como cortas y simples **rutinas** o subprogramas.

En los últimos capítulos se te mostrarán algunas rutinas útiles y cómo sacar provecho de las avanzadas facilidades del MSX. A lo largo del libro hay varias preguntas que deberías intentar contestar (encontrarás las respuestas al final del libro); si las contestas erróneamente, repasa la sección correspondiente hasta que seas capaz de contestarlas correctamente.

El Estándar MSX

El MSX es el nuevo estándar mundial. (Y a la hora de escribir esto, ha sido aceptado por más de 25 grandes compañías de electrónica). Una norma estándar es un enorme paso dentro del mundo de los ordenadores domésticos, y ofrece muchas oportunidades excitantes para el usuario y el programador. El estándar se aplica a la **circuitalia,** al "hardware", tanto como a la **programalia,** al "software", de los equipos.

Supongamos que tienes un Sony MSX y acabas de escribir un programa en él en BASIC o en Código Máquina, y lo has guardado en cinta magnética. Ahora es posible -por la estandarización- coger la cinta y cargarla en otro ordenador MSX. El programa se ejecutará perfectamente sin modificación -ha sido escrito y grabado de acuerdo con el estándar.

Esta intercambiabilidad de los programas es posible debido a la estandarización mencionada. Las variables del sistema están todas en un emplazamiento estándar de la memoria; precisamente la distribución de la memoria y la pantalla es la misma. Así ¿cómo vas a decidirte sobre qué ordenador comprar? Como en el caso de la elección de un sistema Hi-Fi, quizás te guste el aspecto de uno más que de otro, o desees permanecer fiel a una marca en particular. Debes comprar un MSX concreto por sus ventajas y mejoras. Estas mejoras se hacen por fabricantes individuales. Un fabricante puede decidir adaptar a su ordenador la facilidad de acoplar un lápiz óptico. La única cosa a tener en cuenta sobre estas mejoras es que pueden o no estar disponibles en otras máquinas. Si has escrito un programa que sólo funciona con la innovación de un lápiz óptico, es obvio que no funcionará en un MSX que no lo tenga. Y aunque estemos en el tema de mejoras, éstas y periféricos tales como impresoras, también estás gobernados por el estándar.

Desafortunadamente, hay una cosa que no está gobernada por el estándar: la capacidad de la memoria incorporada en el ordenador. Puedes tener 32K, 48K ó 64K dependiendo del fabricante. Un programa funcionará en cualquier MSX con tal que tenga la misma o más memoria que el ordenador en el que fue escrito. Yo personalmente pronostico que la versión 64K rápidamente llegará a ser aceptada como el estándar y las otras "caerán a la cuneta".

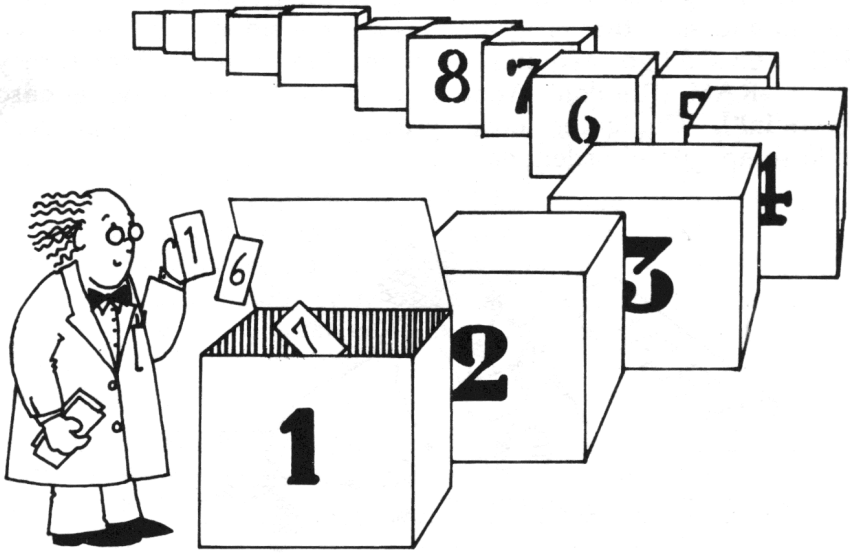
Deberías recordar que MSX es un estándar mundial. Así, por esa razón, si has escrito un programa en Inglaterra, funcionará correctamente también en Japón, España o cualquier otra parte del mundo.

¿Qué es Código Máquina?

¿Qué es Código Máquina?

Es probable que la mayoría de vosotros tenga un ordenador de 64K. K es una abreviación de 'kilo' -y los kilos en informática tienen un poco más de 1000 cosas, exactamente 1024- así 64K significa 65536. Un ordenador de 64K tiene 65536 **lugares** identificados en su memoria 'celdillas', si lo asimilamos a un panal, cada lugar, cada posición puede ser considerada como un cajetín, cada cajetín **rotulado** de 0 a 65535 para distinguirlo. Estos rótulos no existen realmente en ningún sitio, pero podemos referirnos a la primera caja como la 0, a la siguiente como la 1, etc. Cuando primero enciendes el ordenador, tendrás libres más de 28000 posiciones de memoria, las cuales se usarán para programas en BASIC o Código Máquina. Eso es lo que queda de 32768 (32K) después de usar unas 4000 variables del sistema. En un ordenador de 64K, todavía habrá 32K de memoria, en adición a las técnicas de programación en Código Máquina, si quieres usar estas 32K extra -y esto va más allá del propósito de este libro. Todos los programas de este libro funcionarán en un ordenador MSX de 32K (o mayor).

Cada una de esas 'celdillas' de memoria puede contener un número entre 0 y 255 inclusive. Sólo es capaz de almacenar un número de valor 255 como máximo; y eso es obviamente una limitación, por lo que hay que encontrar un método para almacenar números mayores. El siguiente ejemplo muestra cómo se hace.



Digamos que el número que deseas almacenar en memoria es 29248. Primero dividimos ese número entre 256, luego se aproxima la respuesta al número entero más cercano. Por tanto: 29248/256 = 114.25; y aproximado al número entero inferior = 114. Nosotros lo llamamos (114) la **parte alta** del número almacenado. Representa el número total de 256-enas que hay en dicho 29248. La parte alta, 114, en este ejemplo, se multiplicà por 256 y el resultado se resta del número almacenado (29248).

114 x 256 = 29184
29248 - 29148 = 64

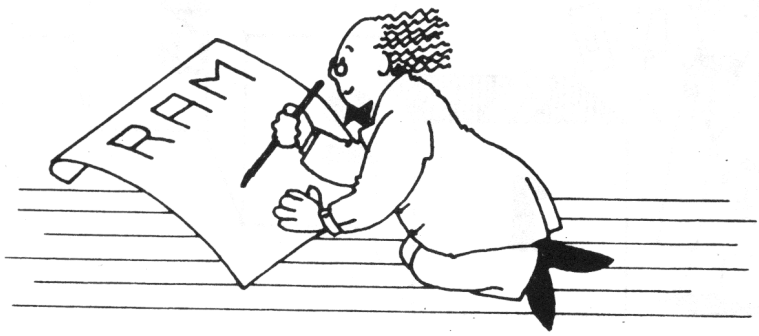
64 se llama **parte baja** del número almacenado.

Para almacenar el número 29248 en memoria, ponemos la parte baja, 64 en una casilla de la memoria y la parte alta, 114, en la **siguiente** casilla. Así pues, si lees en este libro o en cualquier otro, que un número mayor que 255 está almacenado en la casilla 50.000, entenderás que, de hecho, lo almacena en las casillas 50.000 y 50.001.

¿Cuáles son las partes alta y baja del número 45621?

Si la parte baja del número es 31 y la alta 64 ¿cuál es el número?

Números entre 0 y 255 a menudo están referidos como **bytes**, y más modernamente como **octetos** (ya que $2^8 = 256$). Así, si lees que un programa tiene 5000 bytes de longitud, significa que ocupa 5000 lugares-celdillas de memoria. El área de memoria donde almacenas tus programas -sin considerar si están en BASIC o en Código Máquina- se llama Memoria de Escritura y Lectura -RAM (Random Access Memory=Memoria de acceso **'referencial'**). Cualquier cosa grabada en RAM se perderá cuando se apague el ordenador.



Existe, sin embargo, otra área de memoria de Unicamente Lectura abreviada ROM (Read Only Memory). Como su nombre indica, sólo puedes leer lo **contenido** en las casillas que fue escrito por el fabricante. Los contenidos de la memoria ROM no se perderán cuando el ordenador se apague. Esto es precisamente porque contiene el **sistema operativo** -ya que sin él el ordenador sólo sería un inútil trasto complejo de circuitos y transistores. El sistema operativo contiene muchas rutinas en Código Máquina, las cuales llevan a cabo las funciones del ordenador, tales como lectura del teclado, hacer cálculos y efectuar la revisión de la sintaxis en programas en BASIC.

En el centro de tu ordenador hay un circuito **procesador** llamado Z80, y es la parte que realiza el trabajo más duro. No es un cerebro; no es inteligente. De hecho, sólo puede hacer una aritmética muy simple. Pero la ventaja sobre una persona es que puede hacer muchas operaciones por segundo, y por eso parece que es inteligente.

El procesador sólo es capaz de entender instrucciones en Código Máquina, conocidas como **Código de Operación** y abreviadas aquí **'codop'** y en inglés, normalmente "opcode". Hay más de 600 instrucciones en el **repertorio** del Z80. Cada operación está represeentada por su propio número o combinación de números. El número 198 representa la operación de sumar dos números. Así cuando al procesador se le presenta el número 198, sabe que lo siguiente que debe hacer es la suma de dos números (se verá claro más tarde, cómo sabe cuáles son los números que tiene que sumar).

Pero si el procesador sólo puede entender programas en Código Máquina, ¿cómo puede arreglárselas con programas en BASIC? El secreto radica en lo que puede ser considerado como un **interpretador.**
Si un francés está hablando contigo en su lengua nativa, y tú sólo entiendes el español, necesitarás la ayuda de un intérprete. El francés le hablará al intérprete y éste te lo traducirá.

Casi lo mismo ocurre entre un programa en BASIC y el procesador. El programa en BASIC conversa con un 'intérprete' y éste le habla al procesador. Pero aquí el intérprete es en realidad un programa interpretador, escrito obviamente en Código Máquina e incorporado de forma permanente e indelebre en la memoria del propio ordenador. ¡Es lo que el fabricante ha **escrito** en la ROM! Este uso de un interpretador, de un intermediario, lleva más tiempo del que supone tener una "conversacion", ya sea entre un francés y un español, ya entre un programa en BASIC y el procesador Z80 de tu máquina.

Si escribimos nuestros programas en Código Máquina, seremos capaces de cargarlos directamente en la memoria de Escritura/Lectura y de pedir al 'intérprete' que se quite de en medio. "Hablando" directamente con el procesador habrá un considerable incremento en velocidad en la ejecución del programa. En general, un programa en Código Máquina, es de 50 a 100 veces más rápido que el programa equivalente en BASIC. Claro que es a costa de que en la escritura del programa no sea tan extraordinariamente fácil como es en BASIC.

Equivalentes en Código Máquina de las Instrucciones en BASIC

Equivalentes en Código Máquina de las Instrucciones en BASIC

En esta sección describiré que "opcodes" pueden usarse para simular instrucciones en BASIC. Muchas instrucciones en BASIC, tales como LIST, NEW, RENUM, DELETE y AUTO, no son válidas para el propósito de la programación en Código Máquina, así pues no mostraremos ningún **código de operación** para ellas. Algunas instrucciones BASIC tales como RUN, READ, DATA, PRINT, VPOKE, VPEEK, no tienen equivalentes cercanas en Código Máquina pero describiré cómo simularlas y cuándo se necesitan.

Los opcodes que describiremos aquí, son los más comúnmente usados. Son suficientes para permitirte escribir programas simples en Código Máquina.

En programación en BASIC, estarás acostumbrado a usar variables tales como A, B, C... X, Y, Z. En programación en Código Máquina, no hay variables así; sus equivalentes más próximas son los **registros.** Hay muy pocos registros y los que usarás principalmente son:

A,B,C,D,E,H,L

Hay otro registro llamado F pero no nos concierne de momento. Cada registro puede ser considerado como similar a una celdilla de memoria; sólo puede ser almacenado un número de 0 a 255. Para permitirnos usar los registros para almacenar números mayores que 255, seis de esos registros pueden agruparse en 3 pares, como sigue a continuación:

HL
BC
DE

Este emparejamiento no impide que los registros sean usados individualmente. Estarás familiarizado con el siguiente comando BASIC:

```
LET A = 5
```

La operación equivalente cuando se trata de C.M. (la normal abreviatura de Código Máquina) se representa **nemotécnicamente** por:

```
LD A,5
```

LD es una abreviación de la palabra LOAD. Por tanto, lo podríamos expresar como "CARGUE el registro A con el valor 5".

Cualquiera de los otros seis registros, puedes utilizarse con un número entre 0 y 255, de manera similar al registro A.

LD H,199(CARGUE el registro H con el valor 199)
LD D,2(CARGUE el registro D con el valor 2)

Ya he mencionado que cada registro individual puede ser considerado como una celdilla de memoria. Por tanto, recordando cómo son almacenados los números, sabemos que la **parte alta** de 827 es 3 (dividido entre 256 y aproximado al entero inferior), y la **parte baja** es 59. Cuando mandamos que se ejecute la operación LD HL, 827, la parte alta del número se coloca en el registro H y la parte baja en el registro L. (Esto puede recordarse fácilmente pensando en H=High -alta- y L=Low -baja-). En la pareja de registros BC, B es la parte alta. En la pareja DE, D es la parte alta.

BASIC:	LET A = B
Significado:	**Hacer que la variable A tenga el mismo valor que la B.**
Nemotécnico:	LD A,B
Significado:	**Carga el registro sencillo con el valor que haya en el registro B.**

Es posible cargar cualquier registro sencillo con el valor de otro. Puedes mandar, por ejemplo:

LD A,H
LD E,A
LD H,C

El nombre nemotécnico LD (y también hablamos de 'nemónimos' y 'nemónicos' para referirnos a estos nombres y siglas fáciles de recordar y memorizar) es probablemente el más comúnmente usado, y encontrarás difícil escribir un programa en Código Máquina sin él.

BASIC:	LET A = A + 5
Significado:	**Hacer que ahora el valor de A sea igual al que ya había más 5.**
Nemotécnico:	ADD A,5
Significado:	**Añadir (sumar) el valor contenido en el registro A el valor 5.**

El registro A es el único al que puedes añadirle un número directamente. Es decir, no puedes efectuar operaciones tales como:

```
ADD B,9
ADD E,3
```

Esto es, obviamente, una limitación del procesador Z80. Sin embargo, es fácilmente superable, como descubrirás a continuación.

BASIC:	LET A = A + B
Significado:	**Hacer que ahora el valor de A sea igual al que ya había en A más el valor de B.**
Nemotécnico:	ADD A,B
Significado:	**Agregar (sumar) al valor contenido en el registro A el valor contenido en el registro B.**

En esta clase de operación, cualquiera de los siete registros puede ser empleado con el registro A, y el resultado de la suma siempre queda 'acumulado' en dicho registro A, (y de ahí que el registro A se le llame Acumulador)

```
ADD A,B
ADD A,C
ADD A,H
ADD A,L
ADD A,D
ADD A,E
ADD A,A
```

La última de estas operaciones la hemos escrito nemotécnicamente como **ADD A,A.** Esto tiene como efecto **duplicar** el presente valor de A. No es posible sumar cualquier otra combinación de registros diferentes a los mostrados anteriormente. Por tanto no puedes mandar:

```
ADD B,H
ADD D,C
```

Las parejas de registros también pueden intervenir en operaciones de suma en las siguientes combinaciones:

```
ADD HL,BC
ADD HL,DE
ADD HL,HL
```

El resultado de la suma aparecerá siempre en HL. La última operación, ADD HL,HL, obviamente tiene el efecto de duplicar el valor de HL. No es posible sumar el valor de un registro 'simple' directamente a una pareja de registros, o registro doble.

Anteriormente mencioné que no puedes sumar un número directamente a otro registro que no sea A. Ahora te mostraré cómo subsanar esta limitación. Supongamos que el registro B tiene un valor presente de 14 y quieres añadirle 9. El siguiente ejemplo te muestra cómo hacerlo:

Paso 1) **LD A,B**
El registro A ahora tiene el mismo valor que B.
Paso 2) **ADD A,9**
La adición requerida se efectúa y el resultado queda en el registro A.
Paso 3) **LD B,A**
El resultado se transfiere al registro B desde el A.

Con las tres instrucciones anteriores es ahora posible añadir un número al contenido de cualquiera de los siete registros.

Otra limitación del Z80, que ya apunté, era que sólo puedes sumar un registro al registro A. Por tanto no puedes decir:

ADD B,H

Solventaremos esta limitación de la siguiente forma. Digamos que B tiene un valor de 5 y H tiene un valor de 7, y queremos sumar los registros juntos y que el resultado aparezca en el registro B. El siguiente ejemplo te muestra cómo hacerlo:

Paso 1) **LD A,B**
El registro A ahora tiene el mismo valor que B.
Paso 2) **ADD A,H**
La adición es completa con el resultado en el Acumulador.
Paso 3) **LD B,A**
La respuesta se transfiere al registro B desde el A.

Con los tres pasos anteriores es posible sumar cualquier combinación de registros. Puedes incluso sumar el valor de un registro a sí mismo.

Usando pasos similares, podemos sumar cualquier par de registros a cualquier otro par.

Para sumar los valores de BC y DE, con la solución apareciendo en BC, hacemos lo siguiente:

Paso 1) **LD H,B**
LD L,C
El par HL ahora tiene el mismo valor que el BC.
Paso 2) **ADD HL,DE**
La adición se efectúa con el resultado en HL.
Paso 3) **LD B,H**
LD C,L
La respuesta vuelve a transferirse al registro BC.

Salvar las limitaciones del Z80 es un desafío y con práctica aprenderás el modo más eficiente de cumplimentar tus necesidades particulares.

BASIC: LET A = A -5
Significado: **Disminuye el valor actual de A en 5.**
Nemotécnico: SUB A,5
Significado: **Disminuye el valor del registro A en 5.**

Como en la adición, el registro A es el único del que puede restarse un número directamente. Si deseas restar un número de cualquiera de los otros seis registros, debes realizar lo siguiente. Este ejemplo te mostrará cómo restar 5 del registro D.

Paso 1) **LD A,D**
El registro A ahora tiene el mismo valor que D.
Paso 2) **SUB A,5**
La resta se hace con el resultado en el registro A.
Paso 3) **LD D,A**
La respuesta es transferida al D desde el A.

BASIC: LET A = A - B
Significado: **Resta al valor de A el valor de B y lo deja en A.**
Nemotécnico: SUB B
Significado: **Resta del registro A el valor del registro B y lo deja en A.**

De nuevo, el registro A es el único al cual puedes restarle directamente el valor de otro registro.

Para restar el valor de un registro de otro que no sea A, tendrás que usar los tres pasos similares a los que te he mostrado.

Los pares de registros pueden también restarse unos de otros pero sólo en las siguientes combinaciones:

SBC HL,BC
SBC HL,DE
SBC HL,HL

SBC representa una forma especial de sustracción. Por alguna razón oscura, el código de operación (que hubiera sido natural para la sustracción) SUB para pares de registros, no fue usada en el Z80. Explicaré más tarde lo que significa exactamente SBC. Por el momento es suficiente estimarlo como una operación de sustracción ordinaria.

Si simplemente deseas sumar uno a los valores de cualquier registro o registro par, existe un código de operación muy útil llamado INC. Esto es abreviatura de **INCrementar.** Y se usa para **aumentar en uno** el valor del registro que se especifique luego. Las posibles combinaciones son:

INC A
INC B
INC C
INC D
INC E
INC H
INC L
INC HL
INC BC
INC DE

Igualmente, si quieres **disminuir** en uno el valor de cualquier registro, simple o doble, hay una sigla nemotécnica: DEC, que supongo adivinarás es la abreviatura de DECrementar.

BASIC:	**GOTO (número de línea)**
Significado:	**Vaya a la línea de programa numerada así.**
Nemotécnico:	**JP (posición de memoria)**
Significado:	**Salte a esa celdilla de memoria especificada.**

Saltar a la casilla de memoria especificada, es exactamente lo mismo que **ir** al número de línea especificado. Pero un inglés para escribir **ir a...** escribe **go to...,** y para **saltar** escribe **jump** (al menos, los que diseñaron el Z80).

En el siguiente capítulo, explicaré cómo se almacenan en memoria los **códigos de operación,** que corresponden a todos estos 'nombres' **nemotécnicos,** a estos **nemónimos;** entonces todo esto llegará a ser mucho más claro.

BASIC:	GOSUB (número de línea)
Significado:	Vaya a la subrutina que comienza en el número de línea especificado y venga al acabarla. La subrutina se termina con la instrucción RETURN, para que vuelva a donde estaba.
Nemotécnico:	CALL (posición de memoria)
Significado:	La subrutina que comienza en la celdilla de memoria especificada. Como con 'Reclamar' una subrutina en BASIC, las subrutinas en Código Máquina también deben terminarse con una instrucción similar para que vuelva a donde proceda. El vocablo inglés elegido CALL tiene un matiz más fuerte de lo que indica el hispano 'llamar' que algunos usan: es como lo que hace un juez al citar a alguien... inevitablemente se acude a la cita.

BASIC:	IF A= 5 THEN (GOTO/GOSUB/LET/etc.)
Significado:	Si A = 5 entonces haga todo lo que le especifico a continuación.

En la programación en Código máquina, también hay operaciones **condicionadas** al resultado de una cláusula anterior, pero están representadas de modo muy diferente al de BASIC. ¡No existe la instrucción IF como tal! El equivalente de la instrucción BASIC:

IF A = 0 THEN GOSUB número de línea

la escribimos así:

CALL Z, (celdilla de memoria especificada)

Eso significa que si el resultado de la operación más reciente fue Zero, pasaremos a la subrutina que comienza en la celdilla de memoria especificada. Y si no fue Zero, pues no se pasa a dicha celdilla, sino a la siguiente.

Las operaciones **condicionadas** más usadas son:

CALL NZ,nn
Si el último resultado calculado No fue Zero, pasa a la subrutina que empieza en la casilla de memoria especificada 'nn'.

CALL M,nn
Si el último resultado calculado fue negativo -i.e. tiene signo Menos- pasa a la subrutina que comienza en la casilla de memoria 'nn' especificada.

CALL P,nn

Si el último resultado calculado fue positivo -i.e. tiene signo 'Plus'- pasa a la subrutina que comienza en la casilla de memoria nn.

JP Z,nn

Si el último resultado fue Zero, salta a la casilla de memoria nn.

JP NZ,nn

Si el último resultado No fue Zero, salta a la casilla de memoria nn.

JP M,nn

Si el último resultado calculado fue negativo, (i.e. tiene signo Menos) salta a la casilla de memoria nn.

JP P,nn

Si el último resultado calculado fue positivo, (i.e. tiene signo 'Plus') salta a la casilla de memoria nn.

BASIC:	FOR A = 1 TO 100
	Instrucciones que necesitan hacerse 100 veces.
	NEXT A
Significado:	**Repetir lo mandado dentro del bucle, PARA valores de A del 1 AL 100.**
Código Máquina:	LD A,100
	Operaciones que necesitan hacerse 100 veces.
	SUB A,1
	JP NZ, primera operación del bucle.

Para simular un bucle FOR/NEXT primero cargamos el registro A con 100, (o con el número de veces que necesitemos repetir las operaciones que forman el bucle). Al final de cada 'ronda' restamos uno del registro A. Si el resultado de restar uno del registro A fue NZ (No Zero) saltamos al principio de la serie. Así continuamos hasta que finalmente el resultado de restar uno del registro A es Zero, lo cual significa que la serie de operaciones ha sido ejecutada el número de veces requerido.

Cuando programas en BASIC, PEEK y POKE son lo más cerca que llegas del Código Máquina. Esto es, porque estas dos instrucciones son para que se META algo directamente en la memoria, o para mirar lo que HAY dentro de ella:

BASIC:	LET A = PEEK (40000)
Significado:	**Hacer que la variable A tenga un valor igual al que hay contenido en la celdilla de memoria 40000.**
Nemotécnico:	LD A, (40000)
Significado:	**Carga el registro A con el valor contenido en la celdilla de memoria 40000.**

En la posición de memoria 40000 contenía 81, y ejecutamos la operación LD A, (40000), el registro A contendrá ahora 81. El registro A es el único registro simple que puede ser cargado directamente con el contenido de las celdillas de memoria. No puedes, por ejemplo, mandar la siguiente operación:

LD D, (40000)

Pero sí puedes usar cualquiera de las tres parejas de registros, para 'meter' en ellos lo que 'hay' en **dos** celdillas de memoria consecutivas. Así, puedes mandar:

Nemotécnico: LD HL, (40000)

El efecto de esta operación es cargar el registro L con el contenido de la celdilla 40000, y cargar el registro H con el contenido de la 40001. Deberías ser capaz de ver cómo esto es similar a almacenar los números mayores de 255 en celdillas consecutivas de memoria.

Si la posición 40000 de la memoria contiene el valor 5 y la 40001 contiene el 15, ¿cuál será el valor 'total' después de efectuar la operación LD HL, (40001)?

BASIC: POKE (40000),A
Significado: **Coloque el valor de la variable A dentro de la celdilla de memoria 40000.**
Nemotécnico; LD (40000),A
Significado: **Cargue dentro en la celdilla de memoria 40000 el valor que hay en el registro A.**

El registro A es el único registro simple cuyo contenido puede ser cargado directamente dentro de una posición de memoria cualquiera. Pero, cualquiera de las tres parejas de registros puede usarse en una de estas transferencias de datos hasta la memoria, pero -claro- usando dos celdillas de memoria consecutivas. Así:

Nemotécnico: LD (40000),HL
Significado: **Carga la celdilla 40000 con el valor contenido en el registro L y carga la celdilla 40001 con el valor contenido en el registro H.**

Si HL contiene el valor 35621, ¿cuáles serán los valores 'memorizados' en las celdillas 40000 y 40001, después de la operación (40000),HL?

BASIC:	LET A = 5 LET B = 40000 POKE (B),A
Significado:	**Coloca dentro de la celdilla de memoria señalada por el valor de B, el valor contenido en A.**
Nemotécnico:	LD A,5 LD (HL),A
Significado:	**Coloca dentro de la celdilla de memoria señalada por el contenido de la pareja HL, el valor que hay en el registro A.**

BASIC:	LET B = 40000 LET A = PEEK(B)
Significado:	**Haga que la variable A tenga el valor contenido en la celdilla de memoria señalada por el valor de B.**
Nemotécnico:	LD HL,40000 LD A,(HL)
Significado:	**Cargar el registro A con el valor contenido en la celdilla de memoria señalada por el contenido de la pareja HL.**

El registro A es el único registro simple que puede cargarse directamente con el valor que hay en la celdilla de memoria señalada por cualquiera de las tres parejas de registros admitidas. Es decir, son válidas las operaciones:

```
LD A,(HL)
LD A,(BC)
LD A,(DE)
```

Los otros seis registros simples que pueden cargarse sólo desde la celdilla señalada mediante la pareja HL, como se muestra a continuación:

```
LD B,(HL)
LD C,(HL)
LD D,(HL)
LD E,(HL)
LD H,(HL)
LD L,(HL)
```

Nemotécnicos: PUSH
POP

Estos dos extraños **nemónimos** no tienen realmente equivalentes en el MSX BASIC. Sin embargo, son muy importantes en la programación en Código Máquina y el siguiente ejemplo te ayudará a explicar para qué se usan.
Pero antes te diré que vienen a ser lo que el inglés usa para las acciones de **empujar** y **'jalar'** (tirar de algo); por ejemplo, para 'poner' y 'quitar' el corcho de las botellas. Supón que tenemos una **rutina** -un 'trozo' de programa- en que se usan los siete registros como se muestra a continuación:

HL
BC
DE
A

En BASIC si quisiéramos ejecutar esta rutina varias veces, usaríamos un bucle FOR/NEXT. He explicado ya cómo simular un bucle FOR/NEXT en Código Máquina, así que ahora vamos a intentarlo.

```
                    LD A,8 (número de veces que se repetirá
                    el bucle)
(comienzo de rutina) HL
                    BC
                    DE
                    A
                    SUB A,1
                    JP NZ, (comienzo de rutina)
                    Y aquí se siga con lo que sea.
```

Estoy seguro que te darás cuenta que este programa puede que no funcione adecuadamente. Es como establecer un bucle FOR/NEXT tal como FOR A = 1 TO 8, y luego usar la variable A en la rutina dentro del bucle, de manera que se fastidie el 'contador' A. Podrían ocurrir cosas muy extrañas; el valor de A controlando el bucle sería destruído por un mal uso de A dentro de la rutina. En BASIC este problema es fácil de evitar porque hay muchas variables que pueden usarse. En la programación en Código Máquina es un problema real y usamos PUSH y POP para solventarlo, poniendo y quitando el valor de A en una **'percha'**, en un depósito de almacenamiento.

Lo siguiente muestra cómo la rutina debería ser reescrita usando PUSH y POP:

```
                    LD A,8
(comienzo de rutina) PUSH A - aquí se dejaría en la 'percha'
                    HL
                    BC
```

```
DE
A
POP A - aquí se cogería de la 'percha'
SUB A,1
JP NZ, (comienzo de rutina)
```

Después de que hayamos cargado el registro A con 8 usamos PUSH. En términos muy simples esto significa que **apilamos** el valor original de A en sitio "seguro", en una parte de la memoria. Esa parte de la memoria que tiene tratamiento especial, se denomina en hispano: rimero, montón, cúmulo, saca, tazón, **percha**, etc; y en inglés sólo "stack" por eso de las "piras" de **estacas** que amontonadas, hacinadas o apiladas previamente a la construcción de una empalizada, se maneja de forma que: el **primer** tronco que se **quita**, fue el **último** que se **puso**.

Podemos hacer que el procesador **ponga en** 0 y que **quite de** la PERCHA los datos que conciernen a cualquiera de los registros dobles; pero no podemos usarlos en registros simples. Ahora puede oir cómo me dices "Pero acabas de mostrarme cómo usar PUSH A, que se refiere a un registro simple". Admito que lo hice, pero fue para evitar confusión. No existe una operación como PUSH A, pero sí puedes utilizar PUSH con un registro doble llamado AF.

F es un registro simple pero especial, ya que no puede usarse como los otros siete registros simples. Aunque antes describí brevemente el registro F, permíteme decir que es enteramente posible escribir programas muy complicados sin hacer referencia al registro F.

Este registro F es principalmente usado por el procesador Z80, como una colección de ocho **testigos** para poder saber cosas sobre los resultados de diversas operaciones. Dependiendo de los valores contenidos en el registro F, el procesador puede decir cosas tales como si la última operación dió Zero, o si tenía el signo 'Plus' o Menos, o si en la operación hubo que llevar algo de una columna a otra. (Recuerda que a veces en las sumas y restas te ves obligado a decir:

"...siete y cinco son doce y me **llevo** una...")

En este "acarreo" a la posición de las unidades de orden superior, la que establece la diferencia entre las operaciones SUB (sustraer) y SBC (sustraer con 'llevada'). Muy sencillamente, SBC significa que el 'testigo de acarreo' del registro F, deberá tenerse en cuenta al hacer la operación de resta. Pienso que a partir de esta descripción del registro F te darás cuenta que no debería ser usado por un programador inexperto -por tanto, no lo mencionaré de nuevo.

Almacenando los "Códigos de Operaciones" en Memoria

Almacenando los "Códigos de Operaciones" en Memoria

A estas alturas ya tendrás un firme conocimiento de cómo se almacenan números en memoria. Sabrás que sólo puedes almacenar números entre 0 y 255 en cualquier celdilla singular de memoria. También he mencionado que el código de operación puede estar representado por un único número de esos o por una combinación de números de esos. En la práctica lo que esto significa es que algunos (dos o más) códigos, pueden ser representados por un número entre 0 y 255, mientras que otros están representados por un número entre 0 y 255, almacenados en celdillas singulares **consecutivas.**

Los números que representan los 'codopes' están almacenados en memoria como cualquier otro número. Algunos códigos de operación necesitan una única celdilla de memoria, otros necesitan dos consecutivas, otros tres, etc.

Si la primera operación de un programa fuese INC A (incrementar A), para escribirla en memoria, primero necesitaríamos saber qué **número** representa al **nemónimo** INC A. De hecho es el 60. Después, necesitaríamos saber dónde debería comenzar el programa en memoria, digamos 40000 (y en BASIC, mandaríamos POKE 40000,60). Así pues, ése sería el lugar donde comienza el programa en memoria. Si la siguiente operación fuese INK HL, necesitaríamos colocar su 'codope', el número 52, en la casilla 40001,52). El programa entero está construido mediante la **serie de 'codopes'**, los cuales representan la secuencia de operaciones que el procesador ha de llevar a cabo. Los dos 'codopes' que acabo de mencionar, sólo tienen un **octeto** de longitud -esto es, pueden quedar almacenados en una sola celdilla de memoria. Sin embargo, algunas operaciones tales como ADD A,5 tienen "codopes" de dos octetos, y necesitan dos celdillas singulares de memoria. En una celdilla se conserva lo que realmente constituye el **código de operación,** i.e. ADD A. En la celdilla singular consecutiva debemos retener el número que deseas sumar a lo que haya en A. Si ADD A,5 fue la primera operación de un programa que empieza en la celdilla 40000, necesitaríamos colocar 198 (el número para ADD A) dentro de la celdilla 40000; y en la celdilla consecutiva 40001 tendríamos que poner el 5. Esa operación ocupa dos celdillas y decimos que la instrucción tiene dos octetos de longitud.

Cuando el procesador empiece a trabajar según ese programa, (te mostraré cómo hacer que trabaje en un momento) el procesador mirará el contenido de la primera celdilla, y verá que el **codope** es 198 y entonces mirará la celdilla 40001 para ver cuánto añadir a A.

¡No creerás que ve ahí "ADD A", ni mucho menos! Cuando le fabricaron -al microprocesador- se las ingeniaron para imbuirle en su mollera circuital que siempre que "viera" en una celdilla un 198 se preparase para añadir a lo que tuviera en el registro A lo que viera en la celdilla. Así que -muy obediente él- mirará. Una vez que ha completado la adición, irá a sumar qué código de operación hay en la celdilla 40002... Y así es de sencillo.

Algunos codopes tienen dos octetos de longitud y además puede que pare especificar la operación se necesite poner detrás uno o dos octetos de **datos,** haciendo que la instrucción total ocupe tres o cuatro celdillas de memoria consecutivas. Y reitero, que estos números entre 0 y 255, que identifican a una **misma** operación se almacenan en celdillas **consecutivas** de la memoria.

Te habrás dado cuenta que a diferencia de las instrucciones en BASIC, las instrucciones en Código Máquina no tienen números de línea. Las instrucciones de los programas en Código Máquina están almacenads directamente una detrás de otra en memoria, y como cada celdilla tiene su propia **dirección,** al procesador le basta reflejar en 'otro' de sus registros internos la dirección de la instrucción que está ejecutando en cada momento. Cuando una instrucción está completa, el procesador cambia el valor de ese otro registro, para reflejar la dirección siguiente. Y acertadamente llamaremos a ese otro registro que 'lleva la cuenta' el "Contador de Programa", y mejor aún **"Registro Seguidor".**

A pesar de la falta de números de línea, es todavía posible tener instrucciones de **vaya...** y de **venga...,** tales como GOTO y GOSUB. En lugar de decirle los números de línea a donde debe saltar, le decimos la dirección de la celdilla de memoria concreta donde debe ir a buscar el siguiente código de operación. Cuando introducimos un programa en Código Máquina no tecleamos los números que corresponden a los códigos de operación y a los datos usando nuestro sistema de numeración **decimal** (en base 10) normal; en lugar de eso, usamos un sistema de numeración (cuya **base es el 16)** llamado HEXADECIMAL (aunque lo lógico sería "décimohexal). Para darte una idea de cómo quedan los números en esta base 16, lo siguiente muestra unos números decimales y sus equivalentes HEX:

DECIMAL	HEX
0	00
9	09
10	0A
15	0F
16	10
255	FF

Se da una tabla completa de números y sus equivalencias en HEX en el apéndice. A través de este libro, donde es necesario evitar confusión, hemos usado **d** y **h** para distinguir entre un número decimal (base 10) y uno HEX (base 16).

8d = 8 decimal
12h = 12 HEX

¿Cuál es el equivalente decimal de E3h?
Si FBh es la parte alta del número y CBh la parte baja, ¿cuál es el número en decimal?

Usa la tabla decimal/HEX al final de este libro para que te ayude con las dos preguntas anteriores.

No te preocupes demasiado sobre el sistema de numeración HEX. Con el uso de la tabla en el apéndice serás capaz de pasar muy fácilmente de uno a otro. Una ventaja del sistema de numeración HEX entre 0 y 255 es que es mucho más "pulcro" que el decimal. Todos los números HEX entre 0 y 255 están representados por dos cifras del sistema (0 al 9, A a F) un número decimal, entre 0 y 255 puede tener una, dos o tres cifras del sistema (0 al 9). Las cifras empleadas en el sistema de base 10 -el **decimal**- se llaman **dígitos** y tienen la forma o tipo siguientes:

dígitos: 0 1 2 3 4 5 6 7 8 9

Obviamente, ese nommbre le viene de los **diez dedos** que tenemos. Las cifras empleadas en el sistema de base 16 -el "decimohexal"- **no** debieran llamarse **dígitos**, sino "dexitos" (o algo similar) y tienen la forma o tipo siguiente:

dexitos: 0 1 2 3 4 5 6 7 8 9 A B C D E F

o bien, para evitar confusiones entre '0' y 'D' y entre '8' y 'B' cuando se escriben a mano o en algunas calculadoras específicas:

dexitos: 0 1 2 3 4 5 6 7 8 9 A b C d E F

Hay varios métodos de actualidad para la inscripción de datos y codopes dentro de la memoria. He escrito un pequeño programa en BASIC, el cual te permitirá inscribir y cotejar programas en Código Máquina muy rápidamente.

Introduce el siguiente programa y compruébalo concienzudamente.

Luego guárdalo en cinta usando:

SAVE "CAS:METDEX"

```
 10 CLEAR200,39999
 15 CLS
 20 LOCATE 0,0
 25 PRINT "Pon 'enclavamiento' de MAYúsculas"
 30 LOCATE 0,4
 35 PRINT "Pulsa I = Ingresar programa"
 40 LOCATE 0,8
 45 PRINT "Pulsa C = Cotejar bloque"
 50 LOCATE 0,12
 55 PRINT "Pulsa T = Totalizar bloque"
 60 LOCATE 0,16
 65 PRINT "Pulsa F = Finalizar trabajo"
 70 A$=INKEY$
 75 IF A$="I"THEN GOTO 185
 80 IF A$="C"THEN GOTO 380
 85 IF A$="T"THEN GOTO 100
 90 IF A$="F"THEN STOP
 95 GOTO 70
100 CLS
105 LOCATE 0,0
110 PRINT "Teclea DIRE COMienzo"
115 INPUT SA
120 LOCATE 0,5
125 PRINT "Teclea DIRE FINalizo"
130 INPUT EA
135 LET D=0
140 FOR C=SA TO EA
145 LET D=D+PEEK(C)
150 NEXT C
155 CLS
160 PRINT "RECUENTO TOTAL BLOQUE = ";D
165 LOCATE 0,20
170 PRINT "Pulsa W = Vuelva a MENU"
175 IF INKEY$     "W"THEN GOTO 175
180 GOTO 15
185 CLS
190 LOCATE 0,0
195 PRINT "Pon 'enclavamiento' de MAYúsculas"
200 LOCATE 0,4
205 PRINT "Teclea DIREcción COMienzo"
210 INPUT S
215 IF S   40000 THEN GOTO 370
220 LET A$=""
```

```
225 LOCATE 0,23
230 LET ET = S
235 IF A$=""THEN INPUT A$
240 LET MAL$="FALSO"
245 IF A$="W"THEN GOTO 15
250 LET E=LEN(A$)
255 LET E=E-1
260 LET C$=A$
265 FOR D = 1 TO E STEP 2
270 LET B$=LEFT$(C$,3)
275 LET C=VAL("&H"+B$)
280 IF C=0 THEN GOSUB 360
285 LET C$=MID$(C$,3)
290 NEXT D
295 IF MAL$ = "CERTO" THEN GOTO 345
300 LOCATE 0,21:PRINT S;" ";A$
305 LET B$=LEFT$(A$,2)
310 IF LEN(B$)=1 THEN GOTO345
315 LET C = VAL("&H"+B$)
320 POKE(S),C
325 LET S=S+1
330 LET A$=MID$(A$,3)
335 IF A$=""THEN GOTO 230
340 GOTO 305
345 LOCATE 0,22:PRINT"Dato erróneo. Inténtalo otra vez"
350 LET S=ET
355 GOTO 220
360 IF B$     "00"THEN LET MAL$ = "CERTO"
365 RETURN
370 PRINT "La DIRE_COM ha de ser mayor o igual a 40000"
375 GOTO 205
380 LET ND=0
385 CLS
390 LOCATE 0,0
395 PRINT "Teclea DIREcción COMienzo"
400 INPUT SA
405 LOCATE 0,5
410 PRINT "Teclea DIREcción FINalizo"
415 INPUT EA
420 CLS
425 IF SA+20   EA THEN GOTO490
430 FOR C=SA TO SA + 20
435 IF PEEK(C)   16 THEN GOTO480
440 PRINT C;"";HEX$(PEEK(C))
445 NEXT C
450 IF ND=1 THEN GOTO 505
455 IF C   EA THEN GOTO 505
460 PRINT "Pulsa S = Siga mostrando"
```

```
465 LET SA=C
470 IF INKEY$ <> "S" THEN GOTO 470
475 GOTO425
480 PRINT C;"0";HEX$(PEEK(C))
485 GOTO 445
490 FOR C=SA TO EA
495 LET ND=1
500 GOTO 435
505 PRINT "Pulsa W = Vuelva al menú"
510 IF INKEY$ <> "W" THEN GOTO 510
515 GOTO 15
```

Cuando desees trabajar con el programa, basta teclear lo siguiente:

LOAD "CAS:",R

Cuando esté cargado se ejecutará automáticamente (RUN) y verás el siguiente menú:

Pulsa I para Ingresar programa
Pulsa C para Cotejar bloque
Pulsa T para Totalizar bloque
Pulsa F para Finalizar trabajo

Para inscribir en memoria tu programa en Código Máquina debes pulsar la tecla I. Después te avisará que debes usar siempre las mayúsculas (CAPS LOCK 'enclava' el teclado a mayúsculas). Aquí está el primer programa en Código Máquina para que lo metas en la memoria. Sumará dos números y pondrá el resultado en la casilla de memoria 40100. Realmente es sólo una comprobación para asegurarse que has introducido correctamente el programa METDEX y hacer que te familiarices con su uso. La secuencia de acciones correcta para introducir este programa es la siguiente:

0) Pon en marcha el ordenador y con el programa METDEX trabajando.
1) Pulsa la tecla I
2) Aprieta CAPS LOCK (enclave de mayúsculas).
3) Te preguntará la dirección de **comienzo** del programa, en este caso es 40000.
4) Puedes entonces comenzar a meter los códigos HEX. La primera línea es 3E05 (pulsa RETURN cuando **'ya vale'**). Hay dos octetos que van a las casillas 40000 y 40001.
5) Introduce las otras tres líneas de forma similar, pulsando ENTER para indicar la conclusión de una línea.

6) Después de haber introducido la última línea y pulsado RETURN necesitarás volver al menú de opciones pulsando la letra W (de ¡vuelva!).

```
DIREccion COMienzo ..... 40000
DIREcción FINalizo ..... 40007
Recuento TOTAL bloque ..... 852
```

```
3E05    LD A,5
C610    ADD A,16
32A49C  LD (40100),A
C9      RET
```

Lo siguiente que debes hacer es comprobar que has introducido el código correctamente. Se hace pulsando C. Entonces te preguntará la dirección de comienzo del bloque de celdillas que deseas cotejar, en este caso es 40000. Después te pedirá que introduzcas la dirección final del bloque, en este caso es 40007. La pantalla te mostrará entonces las direcciones y los contenidos de cada celdilla de memoria del bloque. Deberías revisar todo cuidadosamente; si hay un error deberías volver al menú y volver a introducir el programa.

Hay otra facilidad de comprobación que se obtiene pulsando la tecla T. Entonces también pedirá la dirección de comienzo y de final del bloque. El programa METDEX sumará los contenidos de todas estas celdillas y expondrá el total; en este caso sería 852. Si no es así, es que has introducido el código incorrectamente o, menos probable, que hayas cometido un error al teclear el programa METDEX. Asegúrate que solucionas el problema antes de proseguir. Todas estas comprobaciones son de vital importancia porque, a diferencia de BASIC -el cual se detiene si tiene un error- un error en Código Máquina normalmente ocasionará que el ordenador se enganche a sí mismo en un bucle sin salida y tendrás que apagarlo. Para probar el pequeño programa que acabas de introducir, primero haz que pare de ejecutar METDEX pulsando la tecla F. Ahora inscribe las siguientes líneas de BASIC:

```
1000 defusr = 40000
1005 a = usr(1)
```

Estas dos líneas pueden ser consideradas realmente como equivalente a un 'VAya Y venGA' (GOSUB) a la rutina en Código Máquina, colocada a partir de la celdilla de memoria 40000. Si miras la última instrucción del programa en Código Máquina, encontrarás el nemónimo RET, que significa RETorne (es decir, que VUELVA a donde corresponde).

Cuando el procesador observe este código de operación regresará al lugar de donde vino, en este caso, vuelve a BASIC. Así es exactamente como funciona una **rutina** en BASIC y en Código Máquina. Ahora basta teclear GOTO 1000 para que se ejecute la rutina en Código Máquina cuya dirección de comienzo se DEFine en la línea 1000, y cuya 'activación' o entrada en acción se logra al **citarla** en la 1005. Ahora teclea PRINT PEEK (40100) para que te EXPOnga lo que HAY en la celdilla con esa dirección dada. El número 21 debiera aparecer en pantalla, ya que es el resultado de sumar 16 y 5.

El proceso anterior para la carga y comprobación de las rutinas en Código Máquina puede aplicarse a todos los listados en este libro, por eso no lo explicaré de nuevo. Es importante fijarse en las direcciones de comienzo y final y en el valor total, y las doy al comienzo de cada listado. El procedimiento para comprobación de rutinas en cada caso es también similar. Se te darán algunas líneas de BASIC para que las introduzcas y siempre empezarán con la línea 1000. Cuando la comprobación de las rutinas en Código Máquina sea satisfactoria, puedes hacer que vuelva a ejecutarse el programa METDEX tecleando RUN.

Entendiendo la Imagen en Pantalla

Entendiendo la Imagen en Pantalla

Una característica muy útil del MSX es sus cuatro diferentes modos de pantalla. Estos son:

MODO 0 24 FILAS 40 COLUMNAS
(960 cuadratines de 8x6 puntos)
MODO 1 24 FILAS 32 COLUMNAS
(768 cuadratines de 8x8 puntos)
MODO 2 MODO DE ALTA RESOLUCION
6144 rayas de 8x1 motitas
(192x256 = 49152 puntos)
MODO 3 MODO MULTICOLOR
64x48 motas de 4x4 puntos,
16 colores por mota

Cada modo tiene sus propias ventajas y debe adaptarse de acuerdo con las necesidades del programa.

A lo largo de este libro trataré exclusivamente con el MODO DE PANTALLA 1, y a no ser que explícitamente mencione lo contrario, todo estará referido al MODO 1. En un capítulo posterior describiré el resto de los modos con mayor detalle. El modo 1 es probablemente el mejor cuando se escriben programas de juegos.

La primera pregunta que surge es "¿Cómo aparece una imagen en pantalla?" Los motivos visuales y el texto que deseas exponer en pantalla son retenidos en memoria como series de números con valor entre 0 y 255. Los circuitos del ordenador están constantemente mirando el **área de memoria** que -como si fuera un "cliché"- contiene la imagen que ha de proyectarse y después hacen el duro trabajo de adaptarla y traspasarla al televisor.

Por tanto, ¿dónde está almacenada la imagen en memoria? Ya he descrito el área de **memoria** de usuario llamada RAM donde se almacenan tus programas en BASIC o en Código Máquina. En algún lugar interno de tu aparato MSX hay otro área de memoria, que llamamos VIDEO RAM (VRAM). Esa RAM tiene 16384 celdillas de un octeto de longitud, en todos los ordenadores MSX.

La VRAM se usa para almacenar la imagen que tú ves en pantalla además de otras cosas conectadas con ella, tales como el **tipo** o forma de cada **carácter** (i.e. letra, cifra, signo de puntuación o gráfico) y el **tipo** o forma de cada **figura** o **motivo** visual que tú definas (y que es obvio no tienen nada de 'espíritus' a pesar de la palabra clave SPRITES usada con el MSX-BASIC, como ya veremos).

En el modo 1 de pantalla hay un mapa en VRAN que tiene 32 celdillas a lo ancho y 24 a lo largo, haciendo un total de 768 celdillas. Cada una de las 768 **direcciones** de las celdillas de ese mapa corresponde a cada una de las **posiciones** de los 768 cuadratines de la pantalla donde es posible exponer un carácter. El número 65 representa el código en BASIC de la letra A; si la primera celdilla de ese mapa contuviese 65, la letra A aparecería en la esquina superior izquierda de la pantalla. Vamos a ensayar y a ver qué ocurre. Lo primero que hay que hacer es encontrar la dirección de la primera celdilla de ese 'mapa' de la memoria de vídeo. El propio ordenador te lo dice si le mandas que:

```
SCREEN 1
PRINT BASE (5)
```

Con el primer comando le obligas a que **adopte** el modo 1. Con el segundo a que **exponga** el número que es la dirección de comienzo de ese mapa de pantalla. En realidad, ese 'mapa' de que hablo es una **'ristra'** de 768 celdillas consecutivas de la VRAM; y siempre se considera como **sede** (o BASE) de un bloque de celdillas contiguas la **dirección** de la **primera** celdilla del bloque.

Deberías anotar este número que sale. Nos referimos a él como PIZACOM. En MSX el comienzo de la PIZArra en modo 1 es 6144. Ahora vamos a escribir en esta pizarra la letra A, es decir, a meter dentro de esta celdilla de la memoria de vídeo el número 65, para lo que basta mandar al ordenador que:

```
VPOKE PIZACOM,65
```

Deberías ver ahora la letra A en la esquina superior izquierda. Es una sencilla cuestión calcular la **dirección** que corresponde a cualquier cuadratín que ocupe otra posición en pantalla. A la esquina superior derecha de la pantalla, por ejemplo, le tocará PIZACOM + 31. (NB: en algunos televisores no es posible ver los extremos de las esquinas de la pantalla, así pues ensaye con otros valores).

Como en todas las celdillas de memoria del MSX de la clase que sea, cabe un sólo **octeto**, sólo podemos meter en ellas un número cualquiera que valga entre 0 y 255; si es en esa franja de la memoria de video, el carácter correspondiente aparecerá en pantalla. La forma o **tipo** de los caracteres también están almacenados -en el modo 1- en otro 'mapa' de la VRAM, cada carácter necesita 8 **rayas** horizontales de 8 puntos con 'uno entre dos' colores para que se sepa como ha de rellenarse el cuadratín que ocupa al ser expuesto en la pantalla.

Se necesitan por tanto 8 celdillas de 8 **bits** para **almacenar su** 'tipo característico'; lo cual hace un **total de 2048** celdillas=octetos para almacenar los tipos **de todos los 256** caracteres del repertorio MSX. También **estas otras 2048** celdillas de la VRAM forman una **ristra (son contiguas)** y podemos hablar de 'mapas' y de 'sedes'. Y **también el ordenador** nos dice la direcciónn donde comienza, si le mandamos que:

PRINT BASE (7)

Esto expondrá un número que representa esa dirección donde comienza en la VRAM el 'mapa' que describe el tipo de los caracteres del repertorio. En mi MSX esa dirección es la cero. Me referiré a ella como TIPOCOM. La dirección de comienzo de las OCHO celdillas consecutivas que describen el tipo o forma de un carácter cualquiera, viene dada por la fórmula.

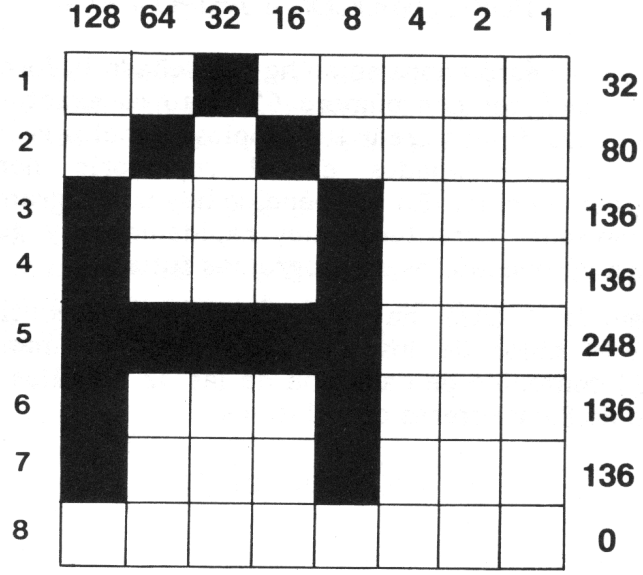
TIPOCOM + 8 * **(número adscrito al carácter)**

Ahora vamos a examinar como el **tipo** o forma de un carácter está almacenado en la VRAM. Usaré la letra mayúscula A como un ejemplo. En mi MSX, las 8 celdillas para la letra A comienzan en la dirección

TIPOCOM + (8 * 65) = 520

(65 es el **código** es el número adscrito internacionalmente para la letra A.)

Cada uno de los 256 caracteres están como dibujados en un **cuadratín** cuadriculado de 8 * 8, como se muestra a continuación para la letra A.



Cada una de las ocho rayas "moteadas" horizontales, representa una de las ocho celdillas que se necesitan para almacenar el tipo característico de la letra A. Cada una de esas rayas puede convertirse en un número. (Si deseas saber cómo se hace, lee la sección en el apéndice sobre el sistema de numeración **binario**). Son estos ocho números los que están almacenados en la VRAM en la ristra de celdillas contiguas reservadas para TIPOS. El siguiente ejemplo muestra cómo están almacenados los tipos según el **prescrito** en fábrica y cómo puedes alterarlo. Vamos allá: Teclea:

```
SCREEN 1
LET PIZACOM = BASE (5)
LET TIPOCOM = BASE (7)
CLS
VPOKE (PIZACOM + 412),65
```

Así expondrás la letra A en la pantalla, en el cuadratín que ocupa la posición 412 de la misma. Lo que haré ahora es alterar realmente el **tipo** que el ordenador da a ese **carácter** número 65, Introduce las siguientes líneas y observa el **tipo** que el 'carácter' expuesto en pantalla va adquiriendo a medida que le demos el 'tratamiento':

```
LET ATIPO = 8 * ASC ("A")
VPOKE (TIPOCOM + ATIPO + 0), 24
VPOKE (TIPOCOM + ATIPO + 1), 36
VPOKE (TIPOCOM + ATIPO + 2), 66
VPOKE (TIPOCOM + ATIPO + 3), 129
VPOKE (TIPOCOM + ATIPO + 4), 255
VPOKE (TIPOCOM + ATIPO + 5), 129
VPOKE (TIPOCOM + ATIPO + 6), 0
```

Espero que hagas ésto ahora como he resideñado la forma visiva -el tipo- del carácter con número 65. Esto es exactamente lo que haces cuando deseas crear tus propios símbolos gráficos, los cuales no están reflejados en el repertorio normal de caracteres, establecido. (En el apéndice hay un programa que te permitirá rediseñar caracteres muy rápidamente y **guardarlos** en cinta para aprovecharlos en programas futuros).

También en la VRAM hay 32 celdillas consecutivas que determinan el color de los 256 caracteres de manera muy peculiar:! El contenido de cada una de las 32 celdillas afecta a un conjunto de 8 caracteres correlativos!

Es imposible tener caracteres de diferente color si ambos pertenecen al mismo conjunto de OCHO, así que una vez que has asignado un color para digamos la A, los caracteres de su mismo conjunto i.e. @, A, B, C, D, E, F, G, tendrán el mismo color. El comienzo de la ristra de 32 celdillas relacionadas con el color puede encontrarse sabiendo el número identificativo (el 6) que ese 'mapa' tiene en la VRAM. Basta mandar:

```
PRINT BASE (6)
```

En mi MSX, la dirección es la 8192; nos referimos a esa dirección como COMTIÑE para demostrar cómo la tabla de color controla el color de los subconjuntos de caracteres, teclea el siguiente ejemplo:

```
SCREEN 1
VPOKE (PIZACOM + 412), 65
```

Esto coloca la letra A en esa posición de la pantalla. Alteramos ahora el octeto que hay en la celdilla de color relacionado con el 'equipo' de 8 caracteres en donde 'juega' la letra A. Para calcular cual de las 32 celdillas controla el color del caracter A y de sus compañero, teclea lo siguiente:

```
PRINT (COMTIÑE + <número adscrito al caracter>\8)
```

Y así puedes aprender (por si no lo sabias ya) cómo se hace la división **entera** usando la barra inclinada **invertida.** En mi MSX, la dirección de la celdilla de la VRAM que afecta al color de la letra A es:

```
8192 + 65\8) = 8200
```

En el siguiente ejemplo estaré **metiendo** en la VRAM un valor en la casilla 8200; tú por supuesto, debes VPOKE la dirección que conseguiste en la operación anterior con tu ordenador.

```
VPOKE (8200),241
```

Esto ocasionará que la letra A sea 'teñida' en blanco sobre un fondo negro. E igual también se teñirán las de su equipo, y sólo las de su equipo, como puedes comprobar pulsando en el teclado a discreción.

Luego prueba con:

```
VPOKE (8200),159
```

Esto ocasionará que la letra A (y su equipo) se presente en tinta roja con fondo blanco.

Hay 16 colores posibles y cada uno de ellos se identifica mediante un código numérico, obviamente entre 0 y 15. Tal y como se muestra a continuación:

0 transparente
1 negro
2 verde medio
3 verde claro
4 azul oscuro
5 azul claro
6 rojo oscuro
7 ciano (más o menos 'cadavérico')
8 rojo medio
9 rojo claro
10 amarillo oscuro
11 amarillo claro
12 verde oscuro
13 magenta (algo como sangre púrpura)
14 gris
15 blanco

Ese código es **metido** directamente dentro de la VRAM para lograr los **dos** colores usados al proyectar todo carácter: el de la tinta de la 'pluma' o color **del frente**, y el del tinte de 'papel' o color del **fondo** del cuadratín que ocupa.

Para calcular el que deseas: toma el código para el color de 'pluma' multiplicado por 16 y el código para el color del 'papel'. Como el código para magenta es 13, y el código para el blanco es 15, haciendo la operación 13 * 16 + 15 = 223 el código para **papel** blanco y **pluma** magenta. Y haciendo la operación 15 * 16 + 13 = 253 tendremos un **frente** blanco sobre un **fondo** magenta.

Figuras y Motivos Visivos

Figuras y Motivos Visivos

Una de las características más útiles de MSX es la posibilidad de diseñar **figuras** 'sprites'. ¿Qué son esos **'espíritus'**? Supongamos que estás escribiendo un juego y necesitas mover algo suavemente a lo largo de la pantalla. Si no tuvieses esa facilidad, primero tendrías que exponer un carácter en una posición, luego sobre-exponer un 'blanco' para borrarlo y luego exponer el carácter en la siguiente posición. Este proceso no es sólo tedioso, es también lento y el movimiento del objeto sería "a trompicones".

Con el uso de los 'motivos' visuales es un disfrute el movimiento de objetos alrededor de la pantalla. Imagina que un motivo es como un carácter ordinario: puedes definir su color y su tipo o forma. Uno de esos **motivos** puede ser llevado o 'portado' a cualquier lugar de la pantalla -no está limitado a ser expuesto dentro de los cuadratines que ocupan los caracteres. Estas **figuras** no tienen que ser borradas antes de que puedan moverse. Y mover una figura desde BASIC o desde Código Máquina, sólo implica cambiar el contenido de una celdilla de memoria.

Es posible tener 32 figuras **'portadas'** a la pantalla **simultáneamente** (aunque hay una limitación a esto, sobre la cual trataré más adelante). Estas figuras o motivos, también pueden tener uno de cuatro tamaños, pero no es posible 'portarlos' a la pantalla con tamaños diferentes. En el siguiente ejemplo usaré figuras de talla pequeña, las cuales, de hecho, ocupan el mismo tamaño que los caracteres normales. Los 32 **portores** de figuras están numerados del 0 al 31. Para establecer el tamaño de las figuras teclea lo siguiente:

```
LET VDP(1) = 224
```

Trataré con las otras tres tallas de **"iconos"** en un capítulo posterior. En la VRAM hay un área de memoria donde se conservan datos para saber el tipo o forma de esas "efigies". La dirección de comienzo de este área puede encontrarse tecleando:

```
PRINT BASE (9)
```

En mi MSX la dirección de comienzo es la 14336. Llamaré a esta dirección FACHACOM -COMienzo de tipo o 'FACHA' de las figuras. Como con caracteres normales, las formas de estos "motivos" pequeños, están almacenadas en 8 celdillas consecutivas. En mi MSX, las 8 celdillas con la forma del "guiñol" **número 0** estarán desde 14336 hasta 14343.

Ahora llenaré estas 8 celdillas con octetos para representar un "fantoche" con forma de Invasor del Espacio.

```
VPOKE (FACHACOM + 0),60
VPOKE (FACHACOM + 1),90
VPOKE (FACHACOM + 2),255
VPOKE (FACHACOM + 3),231
VPOKE (FACHACOM + 4),128
VPOKE (FACHACOM + 5),36
VPOKE (FACHACOM + 6),66
VPOKE (FACHACOM + 7),36
```

Aunque estoy convencido que ya te lo han explicado claramente, te recuerdo que hay 2048 celdillas reservadas para el tipo o forma de estos "pulchinelas" de que hablamos; y por tanto si los de este tamaño emplean sólo 8 celdillas, ¡puedes dar forma a **256 "iconos"**! simultáneamente, y usas números identificativos del 0 al 255. Así que ¿por qué hablamos sólo de 32 "marionetas"? Bien. Una cosa es la forma o **tipo** de los "muñecos" que pueden ser 256 diferentes; y otra el mecanismo **portor** de los mismos que 'lleva' a pantalla al que le digan. De esos sólo hay 32 numerados del 0 al 31.

Si continuamos con el "títere" cuyo tipo hemos definido quedará esto más claro.

Es improbable que algo haya aparecido en pantalla; esto es porque no tenemos todavía declarado **dónde** queremos que aparezca el "pelele", ni los dos colores a usar, ni lo más importante, ¿cuál de los 32 posibles **portores** va a moverlo por la pantalla? En la VRAM hay un área de memoria que contiene información sobre la posición corriente y los colores corrientes de los "monigotes". Cada uno de los 32 "llevadores" tiene cuatro variables asociadas a él:

1) **Posición vertical donde llevar la figura.**
2) **Posición horizontal donde portar la figura.**
3) **Número identificativo de la figura 'portada' a pantalla.**
4) **Color de la figura.**

La dirección de comienzo de estos 32 grupos de variables, que son atributos propios de los 'portores' más que de las Figuras que llevan, puede encontrarse tecleando:

```
PRINT BASE (8)
```

En mi MSX la dirección de comienzo es 6912. Llamaré a esto PORTORCOM. La dirección de comienzo de las cuatro variables que corresponden a cada 'portor' puede encontrarse usando obviamente la siguiente operación.

PORTORCOM + 4 * (Número identificativo del Portor)

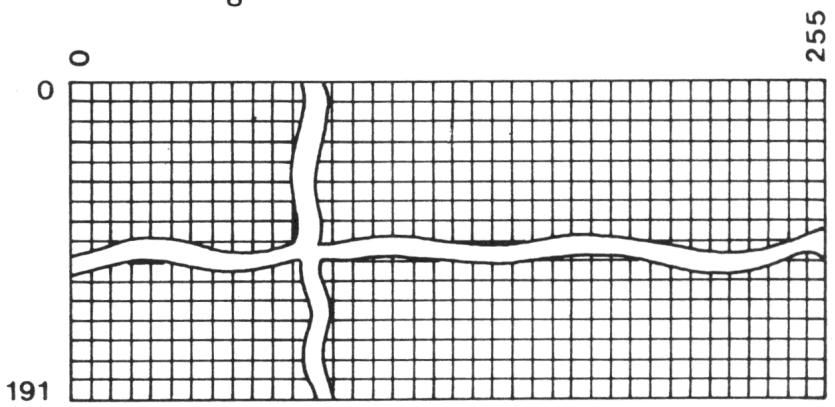
Así en mi MSX el comienzo de las cuatro variables para el mecanismo 'portor' identificado por el número 15, será:

6912 + (15*4) = 6972

Si introduces las cuatro líneas siguientes, harás que sea llevada a pantalla por el Portor número 15, la figura número 0 que hemos diseñado:

```
VPOKE (PORTORCOM + 4 * 15 + 0),100
VPOKE (PORTORCOM + 4 * 15 + 1),50
VPOKE (PORTORCOM + 4 * 15 + 2),0
VPOKE (PORTORCOM + 4 * 15 + 3),10
```

Las coordenadas horizontales y verticales se refieren al lugar donde el punto superior izquierdo de la Figura aparecerá en pantalla. Para el propósito de situar las Figuras, la pantalla está numerada de la siguiente forma:



Intenta meter en la VRAM diferentes valores de las posiciones horizontal y vertical y verás que la figura aparece en otros lugares. Si "punzas" la variable vertical con un número mayor que 191, la Figura desaparecerá de la pantalla. Asociado con las Figuras y los Portores hay un registro especial que puede leerse para ver si dos de las Figuras han chocado. Te mostraré cómo puedes usar esto en el siguiente capítulo en el programa Invasor del Espacio.

Anteriormente mencioné que sólo es posible tener simultáneamente 32 Figuras diferentes en la pantalla. Hay una limitación muy importante a esto que deberías tener siempre presente cuando escribas programas, particularmente programas de juegos. Si tienes más de cuatro Figuras sobre una misma horizontal, la quinta (y las subsiguientes) desaparecerán de esa línea. Esto también se aplica a las Figuras que están sólo "parcialmente" sobre esa línea.

El siguiente diagrama muestra cómo la mitad inferior de la quinta Figura no puede verse:

Esta porción de la quinta figura sobre una misma línea no se verá en pantalla.

El tema entero de las Figuras y sus Portores es muy complejo. Si deseas saber más sobre ellos te aconsejo que compres un libro que trate específicamente con este tema. En el apéndice de este libro hay un programa que te permitirá diseñar Figuras de diversas "fachas" y guardarlas en cinta para uso futuro.

Programa Invasor del Espacio

Programa Invasor del Espacio

Antes de describir el programa Invasor Espacial, me gustaría mencionar una importante característica de los ordenadores MSX llamada Sistema Básico de Entrada/Salida: BIOS. En la memoria ROM está toda la 'legión' de rutinas en Código Máquina que forman el Sistema Operativo. Las puedes usar para facilitar la escritura de programas en Código Máquina. Se dará una descripción completa de las rutinas más importantes en un capítulo posterior. En el programa Invasor Espacial uso algunas de estas rutinas y las describiré brevemente cómo y cuándo sea necesario.

Ahora describiré detalladamente cómo escribir un simple programa: Invasor Espacial. Sin embargo, el hecho de que sea sencillo no significa que los principios implicados en la escritura no puedan ser aplicados para escribir un programa mucho más complicado. Cualquier programa puede ser descrito como una **serie de tareas** a realizar, y cada tarea desglosarse a su vez en un bloque de acciones, y por ende de instrucciones. Al final de este capítulo deberías haber adquirido suficiente conocimiento para ser capaz de comenzar a escribir tus propios programas en Código Máquina.

Empezaré por describir el **diagrama de flujo** para el programa. Si alguna vez has escrito un programa en BASIC no tendrás dificultad en comprender el diagrama de flujo.

Lo siguiente que discutiré es cómo formar un **mapa de memoria** para el programa. No habrás tenido que hacer esto al programar en BASIC, pero es bastante importante cuando empiezas por primera vez a desarrollar tus propios programas en Código Máquina.

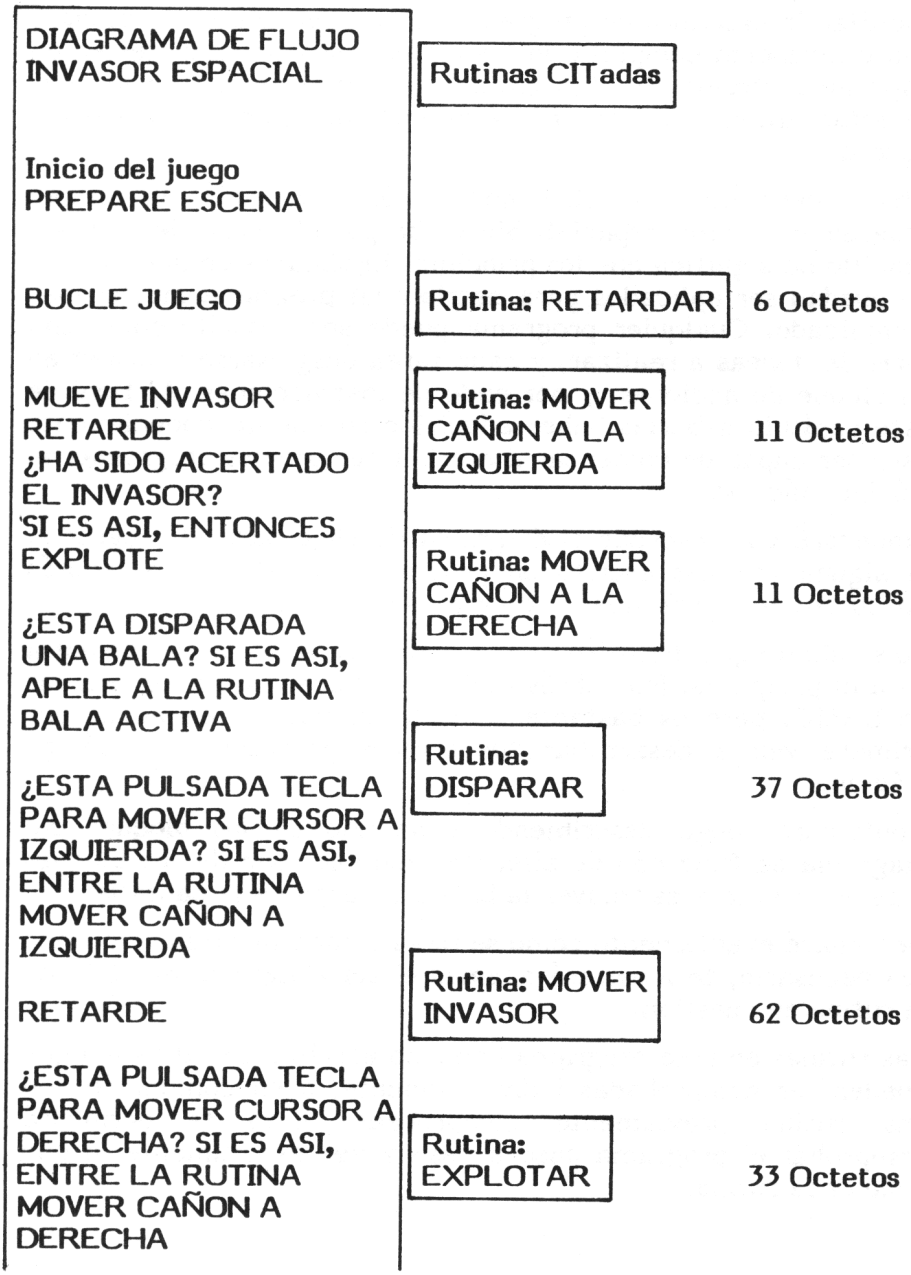
Continuaré luego describiendo cada uno de los **bloques** del diagrama de flujo con detalle. Por ejemplo, uno de los bloques dice que su tarea es "mover la bala a lo largo de la pantalla".

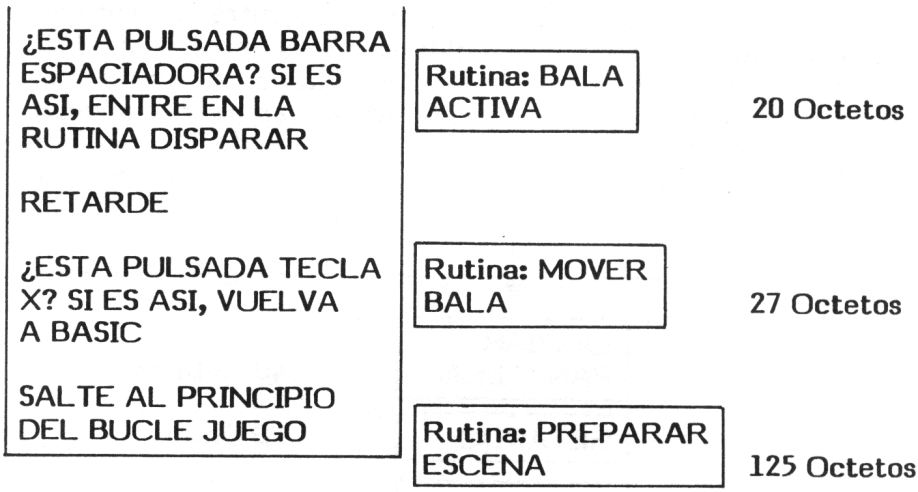
Describiré exactamente cómo se lleva a cabo la tarea, y cuando sea necesario, te mostraré además el **sub-diagrama de flujo** de la rutina en cuestión.

Las rutinas en este programa han sido escritas de tal forma que pueden ser comprobadas individualmente, o en conjunción con una rutina previamente comprobada. Esto te permitirá comprobar el programa paso a paso y ver exactamente lo que hace cada rutina.

EL DIAGRAMA DE FLUJO

Cuando se desarrolla un programa, una de las primeras cosas que debes hacer es dibujar el diagrama de flujo -que en esencia es el **esquema del trabajo** a realizar- que muestra claramente cómo funcionan el programa y las diversas tareas que lleva a cabo. El siguiente diagrama muestra el diagrama de flujo para el programa Invasor Espacial.





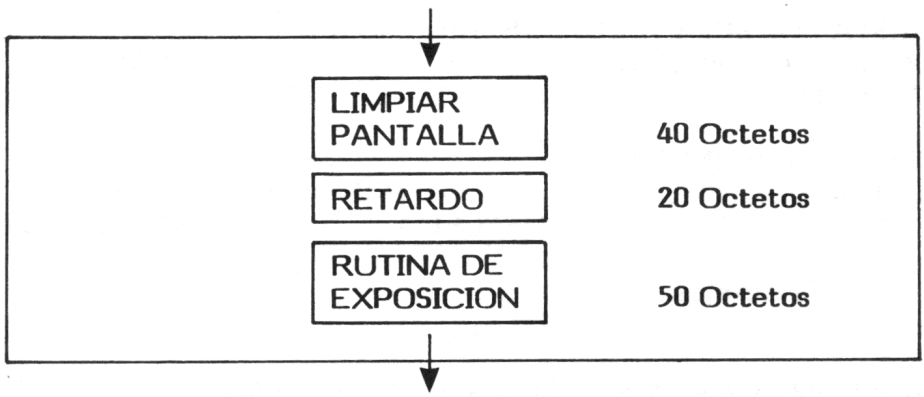
Estoy seguro que estarás de acuerdo en que el diagrama de flujo es relativamente fácil de entender. Deberías estudiarlo cuidadosamente y conseguir un firme entendimiento de cómo funcionará el programa finalizado.

EL MAPA DE MEMORIA

Ya he explicado que los programas en Código Máquina no tienen números de línea. Así pues, si necesitas insertar una nueva instrucción tienes que mover todas las instrucciones que siguen a ésa. Esto entonces ocasiona enormes problemas con **saltos** a otras direcciones y **citas** de subrutinas. Supongamos que tienes una rutina de retardo en la casilla 40000 en el programa. Para **poner en acción** retardo, probablemente tendrías una instrucción en algún sitio que evite esa subrutina, como CALL 40000. Si por alguna razón necesitas añadir una simple instrucción de tres octetos al final de la rutina que termina en la casilla 39999, obviamente necesitarás usar las celdillas 40000/40001/40002. Entonces será necesario mover la rutina de retardo para que empiece en la casilla 40003. Hecho esto, tendrás que trabajar a través de tu programa y cambiar todas esas operaciones, de CALL 40000 a CALL 40003. Escribir un programa de esta forma es tedioso y lleva mucho tiempo.

Estos problemas pueden solventarse usando tu diagrama de flujo principal para crear un **mapa de memoria.** Para formar un mapa de memoria, necesitas saber aproximadamente cuántos octetos ocupa cada rutina.

Entonces repartes las celdillas de memoria entre las rutinas, procurando también dejar unas celdillas libres entre el final de una rutina y el comienzo de la siguiente (...por si acaso). Los espacios de memoria libres te permitirán ampliar cualquier rutina sin que afecte a las siguientes. El siguiente diagrama de flujo -para un programa muy simple- demuestra cómo crear un mapa de memoria:



En el diagrama de flujo, cada una de las tres rutinas está marcada con el número aproximado de octetos que ocupan. La rutina Limpiar Pantalla podría ponerse en la celdilla 40000, la rutina Retardo en la 40050 y la rutina de Exposición en la celdilla 40080. Esto dejará unas celdillas libres entre cada rutina, así cada una puede ser ampliada si se necesita, sin necesidad de reconstruir todo el programa.

Si miras el diagrama de flujo para el programa Invasor Espacial verás que cada rutina está marcada con el número aproximado de octetos que usa. Usando esta información, he creado el siguiente mapa de memoria. La columna de la izquierda muestra el nombre de cada rutina, y la columna de la derecha su dirección de comienzo.

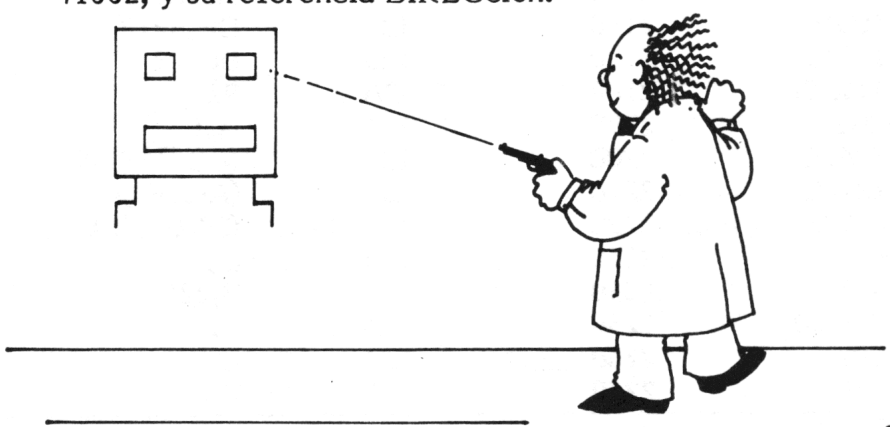
BUCLE JUEGO	40000
BALA ACTIVA	40082
EXPLOTAR	40133
DISPARAR	40157
MOVER BALA	40205
MOVER INVASOR	40243
MOVER CAÑON A IZQUIERDA	40316
MOVER CAÑON A DERECHA	40338
PREPARAR ESCENA	40363
RETARDAR ACCION	40474

En el mapa de memoria también he reservado un área de memoria para las **variables.** (Explicaré el propósito de estas variables en breve).

EXPLICACION DE LOS TERMINOS

Me gustaría explicar unos términos que harán el total de la explicación del juego mucho más fácil:

1) **Bucle Juego:** Esta es la serie de acciones que se **repiten** dentro del programa. Dentro de cada **ronda** del bucle se hacen cosas tales como, inspeccionar si las teclas de mover el cañón a la izquierda, a la derecha o la barra de disparar han sido pulsadas. También repetidamente recurre a la rutina que mueve al Invasor Espacial. Una vez que todas estas funcionnes han sido llevadas a cabo, el programa salta al principio y las hace todas de nuevo, de ahí el nombre de **Bucle** (que nos recuerdan los 'rizos' y 'tirabuzones').
2) **Señalizador de tocado:** Esta es una celdilla de memoria que normalmente contiene el número **uno.** Si la bala alcanza al Invasor, esa casilla es puesta a cero. La celdilla del Señalizador de 'TOCADO' es la 41000, y su referencia DEADFLAG ('Banderín de muerto').
3) **Señalizador de Bala Activa:** Es una única celdilla de memoria que normalmente contiene el número uno. Cuando ya ha sido disparada una bala, esta celdilla es puesta a cero. Se usa para impedir que otra bala sea disparada mientras haya una en pantalla. La celdilla del señalizador de BALA ACTIVA es la 41001, y su referencia ACTFlag (Banderín de 'bala' ACTiva).
4) **Señalizador de Dirección del Invasor:** Esta es una única celdilla de memoria que contiene el número uno si el Invasor se está moviendo de izquierda a derecha. La celdilla del señalizador de dirección del Invasor es la 41002, y su referencia DIRECción.



RUTINA DE INICIACION

La primera rutina que escribiré se llama rutina de iniciación. Debe hacer lo siguiente:

- Establecer el tamaño de Figuras en dos veces al tamaño normal.
- Establecer el COMienzo de la zona para formas de Figuras en la dirección 14336.
- Establecer el COMienzo de la zona VRAM para los Portores en la dirección 6912.
- Establecer los valores iniciales de las cuatro variables de cada una de las tres Figuras que son usadas (son, el Invasor, la Bala y el Cañón disparador).
- Describir la forma o tipo de cada una de esas tres Figuras.

Los 24 octetos que fijan el 'tipito' de las Figuras se colocan en las celdillas 14336 a 14359 mediante esta rutina de iniciación. (El término inglés SETUP que aparece como **rótulo identificativo** de la dirección de memoria 3E01 indica 'preparación completa').

Dirección de comienzo 40363
Dirección de final 40463
Suma total de los octetos 6910

```
3E01      2050 SETUP:  LD     A,1
32AFFC    2060         LD     (64687),A
CD5F00    2070         CALL   95
21EC9D    2080         LD     HL,DATA
110038    2090         LD     DE,14336
011800    2100         LD     BC,24
CD5C00    2110         CALL   92
0E05      2120         LD     C,5
0636      2130         LD     B,54
CD4700    2140         CALL   71
21049E    2150         LD     HL,DATA1
11001B    2160         LD     DE,6912
010C00    2170         LD     BC,12
CD5C00    2180         CALL   92
3E01      2190         LD     A,1
32269E    2200         LD     (DEADF),A
32279E    2210         LD     (ACTF),A
32289E    2220         LD     (DIREC),A
0E01      2230         LD     C,1
06E1      2240         LD     B,225
CD4700    2250         CALL   71
0E02      2260         LD     C,2
0606      2270         LD     B,6
```

```
CD4700      2280          CALL    71
C9          2290          RET
1818187E    2300 DATA:    DEFB    24,24,24,126
FFFFFF00    2310          DEFB    255,255,255,0
3C7E99FF    2320          DEFB    60,126,153,255
663C4224    2330          DEFB    102,60,66,36
18181818    2340          DEFB    24,24,24,24
18181818    2350          DEFB    24,24,24,24
AA64000F    2360 DATA1:   DEFB    170,100,0,15
0000010C    2370          DEFB    0,0,1,12
C8000201    2380          DEFB    200,0,2,1
```

Esta rutina de iniciación superior puede comprobarse mediante las siguientes líneas de BASIC:

```
1000 DEF USR = 40363
1005 A = USR(1)
1010 GOTO 1010
```

Deberás ver al Invasor en la esquina superior izquierda de la pantalla y el cañón disparador en la parte inferior de la pantalla en el medio.

RUTINA PARA MOVER EL CAÑON A LA IZQUIERDA

La siguiente rutina se pone en acción cuando se ha pulsado la tecla correcta para mover el cañón disparador a la izquierda. La rutina comprueba primero si la Cañón está en el extremo izquierdo del borde de la pantalla; si es así, obviamente el cañón no puede moverse más y la rutina vuelve al bucle de juego. Si el disparador no está en el borde del extremo izquierdo **se corre** una posición a la izquierda. Esto se hace **mirando** cuál es la posición horizontal corriente del Cañón, restándole una, y poniendo este nuevo valor dentro de la celdilla de memoria en la VRAM, que corresponde a la posición horizontal del cañón disparador.

Dirección de comienzo 40316
Dirección de final 40327
Suma total de los octetos 1084

```
21011B      1720 MOVLEF   LD      HL,6913
CD4A00      1730          CALL    74
3D          1740          DEC     A
C8          1750          RET     Z
CD4D00      1760          CALL    77
C9          1770          RET
```

La rutina que mueve el cañón disparador a la izquierda puede comprobarse con el siguiente programa en BASIC:

```
1000 DEF USR = 40363
1005 A = USR(1)
1010 DEF USR = 40316
1015 A = USR(1)
1020 FOR B = 1 to 100
1025 NEXT B
1030 GOTO 1015
```

Deberías ver moverse hacia la izquierda de la pantalla tan lejos como puede, el cañón disparador.

RUTINA QUE MUEVE EL DISPARADOR A LA DERECHA

La rutina para mover el disparador a la derecha es muy similar a la que lo mueve a la izquierda. La única diferencia es que se comprueba si el disparador está en el extremo derecho de la pantalla; y segundo, se suma uno -no se resta- a la posición horizontal corriente.

Dirección de comienzo	40338
Dirección de final	40352
Suma total de los octetos	1916

```
21011B  1880 MOVRIG  LD    HL,6913
CD4A00  1890         CALL  74
D6F0    1900         SUB   240
C8      1910         RET   Z
C6F1    1920         ADD   A,241
CD4D00  1930         CALL  77
C9      1940         RET
```

La rutina que mueve el cañón disparador a la derecha puede comprobarse cambiando la línea 1010 en el programa en BASIC anterior para que sea:

```
1010 DEF USR = 40338
```

RUTINA PARA MOVER AL INVASOR

Esta rutina es la más compleja de todas. La siguiente lista muestra qué ocurre cada vez que entra en acción:

1) Si el señalizador de dirección del Invasor está 'alzado', i.e. su valor uno, salta para mover al Invasor de derecha a izquierda.
2) El señalizador de dirección ahora está 'bajado', i.e. su valor es cero; así el Invasor está en ese momento moviéndose de izquierda a derecha.
3) Si el Invasor está en el extremo derecho del borde de la pantalla, se coloca el señalizador de dirección al valor uno y vuelve al bucle de juego.
4) El Invasor no está en el extremo derecho del borde, por eso se mueve una posición a la derecha. El Invasor se mueve de forma muy similar al cañón disparador; es decir, que tomamos su posición horizontal actual, y en el caso de movimiento a la derecha, le añadimos uno. Este nuevo valor se pone dentro del lugar correcto de la VRAM que contiene la posición horizontal de la figura del Invasor.
5) Vuelve al bucle de juego.
6) El señalizador de dirección del Invasor está con valor uno, así el Invasor está moviéndose de derecha a izquierda.
7) Si el Invasor está en el extremo izquierdo de la pantalla, entonces el señalizador de dirección es cero y el programa vuelve al bucle de juego.
8) El Invasor no está en el extremo izquierdo del borde de la pantalla, por eso se mueve una posición a la izquierda. Se hace restando uno de la posición horizontal de esa Figura.
9) Vuelve al bucle de juego.

Dirección de comienzo 40243
Dirección de final 40305
Suma total de los octetos 5797

```
3A289E  1350 MOVINV  LD    A,(DIREC)
3D      1360         DEC   A
CA569D  1370         JP    Z,ILEFT
21051B  1380         LD    HL,6917
CD4A00  1390         CALL  74
D6F0    1400         SUB   240
C24B9D  1410         JP    NZ,IRIG1
3E01    1420         LD    A,1
32289E  1430         LD    (DIREC),A
C9      1440         RET
```

```
21051B    1450 IRIG1:   LD      HL,6917
CD4A00    1460          CALL    74
3C        1470          INC     A
CD4D00    1480          CALL    77
C9        1490          RET
21051B    1500 ILEFT:   LD      HL,6917
CD4A00    1510          CALL    74
D601      1520          SUB     1
C2679D    1530          JP      NZ,ILEF1
3E00      1540          LD      A,0
32289E    1550          LD      (DIREC),A
C9        1560          RET
21051B    1570 ILEF1:   LD      HL,6917
CD4A00    1580          CALL    74
3D        1590          DEC     A
CD4D00    1600          CALL    77
C9        1610          RET
```

La rutina que mueve al Invasor puede comprobarse con las siguientes líneas:

```
1000 DEF USR = 40363
1005 A = USR(0)
1010 DEF USR = 40243
1015 A = USR(0)
1020 GOTO 1015
```

RUTINA DEL BOTON DE DISPARO

En esta rutina se entra cuando está pulsada la barra espaciadora. Lo primero que hay que hacer es comprobar que no hay ya una bala en la pantalla. Si la hay, el programa vuelve al bucle de juego. Esto se comprueba mirando el señalizador de bala activa; si es cero, la bala está en algún lugar de la pantalla -otra no se puede disparar así que el programa vuelve al bucle de juego. Si el señalizador de bala activa está en uno, la bala no ha sido todavía disparada y podemos continuar con el resto de la rutina.

Lo siguiente que se hace es calcular la posición horizontal y vertical de la Figura bala; de manera que la bala parezca salir del extremo del cañón disparador. La posición vertical de la bala será 16 menos que el disparador; es decir, 154.

El señalizador de bala activa está entonces en cero. Esto cumple dos propósitos. Primero, no puede lanzarse otra bala; y segundo, el bucle de juego que debe apelar constantemente a la rutina que mueve la bala (como se describirá más tarde).

Dirección de comienzo	40157
Dirección de final	40194
Suma total de los octetos	2697

```
3A279E   870 FIRE:  LD    A,(ACTF)
3C       880        INC   A
3D       890        DEC   A
C8       900        RET   Z
21011B   910        LD    HL,6913
CD4A00   920        CALL  74
21091B   930        LD    HL,6921
CD4D00   940        CALL  77
21001B   950        LD    HL,6912
CD4100   960        CALL  74
D611     970        SUB   17
21081B   980        LD    HL,6920
CD4D00   990        CALL  77
3E00    1000        LD    A,0
32279E  1010        LD    (ACTF),A
C9      1020        RET
```

La rutina del botón de disparo puede comprobarse con las siguientes líneas. Cuando lo hagas deberás ver aparecer la bala justo encima del disparador:

```
1000 DEF USR = 40363
1005 A = USR(0)
1010 DEF USR = 40157
1015 A = USR(0)
```

RUTINA QUE MUEVE LA BALA

Esta rutina sube la bala una posición en la pantalla cada vez que entra en acción. Esto se hace restando uno del valor actual de la coordenada vertical de la Figura. La rutina también comprueba si la bala ha alcanzado la parte superior de la pantalla. Si es así, el señalizador de bala activa se pone a uno, y la coordenada vertical de la Figura bala se fija a un valor adecuado para que la Figura no pueda verse en pantalla.

Dirección de comienzo 40205
Dirección de final 40232
Suma total de los octetos 2400

```
21081B  1130 MOVBUL  LD    HL,6920
CD4A00  1140         CALL  74
3D      1150         DEC   A
CA1B9D  1160         JP    Z,TOP
CD4D00  1170         CALL  77
C9      1180         RET
3EC8    1190 TOP:    LD    A,200
21081B  1200         LD    HL,6920
CD4D00  1210         CALL  77
3E01    1220         LD    A,1
32279E  1230         LD    (ACTF),A
C9      1240         RET
```

La rutina para mover la bala puede comprobarse con las siguientes líneas. Verás que la bala constantemente se mueve desde abajo hasta arriba por la pantalla.

```
1000 DEF USR = 40363
1015 A = USR(0)
1020 DEF USR = 40157
1025 A = USR(0)
1030 DEF USR = 40205
1035 A = USR(0)
1040 GOTO 1035
```

RUTINA DE BALA ACTIVA

El bucle de juego constantemente comprueba para ver si el señalizador de bala activa está en cero; si es así se ejecutará la siguiente rutina. Lo primero que hace esta rutina es poner en marcha la rutina que mueve la bala, cuatro veces. Esto significa que la bala puede moverse a una velocidad rápida. Entonces comprueba si la bala ha chocado con el Invasor. Esto se hace comprobando un **registro** en la VRAM conocido como Banderín o Testigo de Colisión. Este señalizador de 'choques' está en uno si ninguna de las Figuras choca, por eso siempre que este señalizador esté a cero sabemos que la bala ha golpeado al Invasor. Si el señalizador de colisión está a cero, el señalizador de Tocado se coloca a cero; con lo que indicamos al bucle de juego que el Invasor ha sido alcanzado y que se debe producir la explosión. (Describiré esto más tarde).

Dirección de comienzo	40082
Dirección de final	40102
Suma total de los octetos	2416

```
CD0D9D    420 BULACT   CALL    MOVBUL
CD0D9D    430          CALL    MOVBUL
CD0D9D    440          CALL    MOVBUL
CD3E01    450          CALL    318
CB6F      460          BIT     5,A
C8        470          RET     Z
3E00      480          LD      A,0
32269E    490          LD      (DEADF),A
C9        500          RET
```

RUTINA DE EXPLOSION

El bucle de juego constantemente comprueba el Señalizador de Tocado; si está en cero el programa salta a la rutina de explosión. La forma en que se produce la explosión es simple pero espectacular. Primero, la pantalla pasa al modo 3 -el modo multicolor. El mapa de memoria se llena con números aleatorios que dan una imagen de muchos colores. Esto se repite varias veces, causando un original efecto de explosión.

Dirección de comienzo	40113
Dirección de final	40146
Suma total de los octetos	3587

```
3E03      610 EXPLO:   LD      A,3
32AFFC    620          LD      (64687),A
CD5F00    630          CALL    95
3E64      640          LD      A,100
210000    650          LD      HL,0
E5        660 EXPLO1   PUSH    HL
F5        670          PUXH    AF
110008    680          LD      DE,2048
01E803    690          LD      BC,1000
CD5C00    700          CALL    92
F1        710          POP     AF
E1        720          POP     HL
24        730          INC     H
3D        740          DEC     A
C2BE9C    750          JP      NZ,EXPLO1
C3409C    760          JP      START
```

RUTINA DE RETARDO

Como puede verse a partir del diagrama de flujo, está incluida una rutina para demorar las acciones. Esto se necesita para mover lentamente las Figuras de manera que puedas ver lo que está ocurriendo. El bucle de retardo funciona cargando el registro A con un valor de 255 y disminuyéndolo hasta que se alcance el cero. Cuando el registro A finalmente logra el cero, el programa vuelve al bucle de juego. No es práctico comprobar esta rutina, hasta incluso con el valor máximo de retardo, 255, es todavía demasiado rápido para ser detectada por el promedio humano. Al final de este capítulo demostraré cómo puedes probar que el retardo está realmente funcionando.

Dirección de comienzo 40474
Dirección de final 40480
Suma total de los octetos 959

```
3EFF      2490 DELAY:  LD    A,255
3D        2500 DEL:    DEC   A
C21C9E    2510         JP    NZ,DEL
C9        2520         RET
```

La rutina de explosión se puede comprobar con las siguientes líneas:

```
1000 POKE 40000,201
1005 DEF USR = 40113
1010 = USR(0)
```

EL BUCLE DE JUEGO

Ahora describiré el bucle de juego con gran detalle. Primero, aquí hay una lista de las acciones que se repiten constantemente.

1) Poner en acción a la rutina de mover al Invasor.
2) Mirar si el señalizador de Tocado es cero; si es así salta a la rutina de explosión.
3) Mirar si el señalizador de bala activa es cero; si es así entra en la rutina bala activa.
4) Si la tecla que mueve el cursor a la izquierda está siendo pulsada, apela a la rutina que mueve el cañón a la izquierda.
5) Si la tecla que mueve el cursor a la derecha está siendo pulsada, recurre a la rutina que mueve el cañón a la derecha.

6) Si la barra espaciadora está siendo pulsada, acude a la rutina de botón de disparo.
7) Si la tecla X está siendo pulsada, vuelve a BASIC.

El bucle de juego también "cita" la rutina Retardo varias veces. Esto es para que la ejecución del programa sea más lenta.

Sin esta demora las cosas se moverían tan deprisa que difícilmente serías capaz de apreciarlas.

Las 'apelaciones' a las rutinas no necesitan ninguna explicación. Ahora explicaré cómo es posible detectar qué tecla está siendo pulsada. El programa no hace el equivalente de preguntar "¿Qué tecla está siendo pulsada?" -sino que más bien pregunta "¿Está siendo pulsada esta tecla específica?". En el BIOS hay una rutina que podemos usar para ver si una tecla concreta está siendo pulsada. Para que la rutina funcione tenemos que suministrarle una pequeña información antes de llamarla. Mira el siguiente diagrama; muestra las 9 filas numeradas de 0 a 8 cada una de las cuales contiene 8 teclas:

	7	6	5	4	3	2	1	0
0	7	6	5	4	3	2	1	0
1	;	]	[	\	=	-	9	8
2	B	A	£	/	.	,	'	`
3	J	I	H	G	F	E	D	C
4	R	Q	P	O	N	M	L	K
5	Z	Y	X	W	V	U	T	S
6	F3	F2	F1	CODE	CAP	GRAPH	CTRL	SHIFT
7	RETURN	SEL	BACK SPACE	STOP	TAB	ESC	F5	F4
8	RIGHT CURSOR	DOWN CURSOR	UP CURSOR	LEFT CURSOR	DEL	INS	HOME	SPACE

Supongamos que deseamos detectar si la tecla que mueve el cursor a la izquierda está pulsada, este es el procedimmiento para hacerlo:

Cargue el registro A con 8 (ya que 8 es el **número de la fila** en el diagrama anterior de la que contiene la tecla de mover el cursor a la izquierda).
CALL 321d (Esta es la celdilla donde comienza la rutina que efectúa la lectura de la fila que has especificado).

La rutina ahora pondrá un valor dentro del registro A que guarda relación con cuál de las 8 teclas de dicha línea has pulsado. Si miras el diagrama anterior verás que cada una de las 8 columnas está numerada de 0 a 7. La tecla que mueve el cursor a la izquierda está en la columna marcada con 4.

Para probar si esta tecla ha sido pulsada, usamos la siguiente instrucción:

BIT 4,A

que pondrá a Zero el acumulador sólo si la tecla está pulsada. Luego basta que la siguiente instrucción **cite** la rutina pertinente de forma condicional. Es decir, usaremos:

CALL Z, **(rutina Mover Cañón a la izquierda)**

Por lo tanto, si el resultado de la instrucción BIT 4,A fue Zero (esto es, la tecla estaba pulsada), se acude a la rutina que mueve el cañón disparador a la izquierda. El Código Máquina para el bucle de juego se muestra a continuación.

Dirección de comienzo	40000
Dirección de final	40071
Suma total de los octetos	8681

```
CDAB9D   20 START:  CALL   SETUP
CD339D   30 LOOP:   CALL   MOVINV
CD1A9E   40         CALL   DELAY
3A269E   50         LD     A,(DEADF)
3C       60         INC    A
3D       70         DEC    A
CAB19C   80         JP     Z,EXPLO
3A279E   90         LD     A,(ACTF)
3C       100        INC    A
3D       110        DEC    A
CC929C   120        CALL   Z,BULACT
3E08     130        LD     A,8
CD4101   140        CALL   321
CB67     150        BIT    4,A
CC7C9D   160        CALL   Z,MOVLEF
CD1A9E   170        CALL   DELAY
3E08     180        LD     A,8
CD4101   190        CALL   321
CB7F     200        BIT    7,A
CC929D   210        CALL   Z,MOVRIG
3E08     220        LD     A,8
CD4101   230        CALL   321
CB47     240        BIT    0,A
```

CCDD9C	250	CALL	Z,FIRE
CD1A9E	260	CALL	DELAY
3E05	270	LD	A,5
CD4101	280	CALL	321
CB6F	290	BIT	5,A
C8	300	RET	Z
C3439C	310	JP	LOOP

El programa entero puede comprobarse introduciendo las siguientes líneas:

```
1000 DEF USR = 40000
1005 A = USR(0)
```

Si deseas ver cómo sería el juego sin el Retardo, teclea la siguiente línea, y ejecuta el programa de arriba:

```
POKE 40475,1
```

La celdilla 40475 contenía anteriormente el valor 255; que representa la cantidad de Retardo con que estábamos funcionando.

Facilidades Avanzadas del MSX

Facilidades Avanzadas del MSX

En este capítulo trataré algunos de los temas que hasta ahora no me parecieron necesarios mientras aprendías los fundamentos del Código Máquina. En un libro como éste -sobre un tema tan amplio- resulta muy difícil tratar cada asunto con detalle; si deseas saber más acerca de un concepto o práctica en particular necesitarás adquirir un libro especializado. En muchos casos no es necesario entender totalmente algo antes de poderlo usar. Una de mis frases favoritas es "Si funciona, úsalo"; un gran número de personas conduce coches, por ejemplo, sin la más ligera idea de mecánica. Este es el tema de este capítulo. Debatiré algunas de las avanzadas características del MSX y cómo usarlas, pero no describiré detalladamente cómo funcionan.

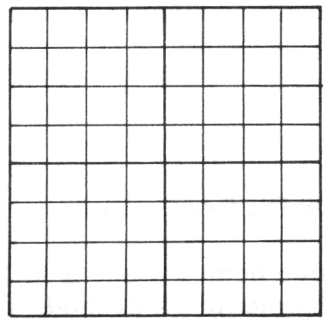
LOS CUATRO MODOS DE PANTALLA

El modo de pantalla que hemos usado a lo largo de este libro es el MODO 1, y probablemente es el modo más útil para usar en juego del tipo de "Comecocos" o "Tiramuros".

En el MODO 0 de pantalla puedes tener 24 filas, cada una conteniendo 40 **'cuasi**-cuadratines'. Teclea SCREEN 0 para ver este modo. En este modo es posible tener sólo dos colores, el color de la pluma o frente y del papel o fondo. El reborde -el "limbo"- tomará el mismo color que el papel. Este modo es el más usado para programas de texto tales como aplicaciones para componer documentos. No es posible 'portar' figuras en el MODO 0.

El MODO 2 de pantalla es muy similar al MODO 1. Dispone de 768 cuadratines 8 * 8, y cada uno puede presentar una forma totalmente diferente. Además, cada una de las ocho **"rayas moteadas"** horizontales dentro de cada cuadratín, puede utilizar la **pareja** de colores propia con entera libertad.

El MODO 3 de pantalla se conoce como modo multicolor y es extremadamente complejo. Como en el MODO 1, la pantalla se divide en 24 filas conteniendo 32 cuadratines cada una, dando 768 posiciones en total. En lugar de exponer caracteres dentro de cada uno de estos cuadratines 8 * 8, los usa como si estuvieran sub-divididos en cuatro **motas** de 4 * 4 **puntos** cada una.



Y a cada mota de 4 * 4 puntos puede dársele cualquier color de los 16 disponibles.

MOTIVOS VISIVOS O FIGURAS PORTABLES

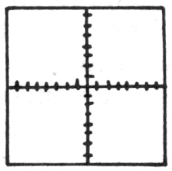
En un capítulo anterior, sólo una de las **tablas** de las Figuras fue tratada con detalle. Ahora te mostraré las otras tres tallas de 'sombras chinescas' que puedes usar. Primero ponemos una Figura de tamaño normal en la pantalla, tecleando lo siguiente:

```
SCREEN 1
VDP(1) = 224
VPOKE (PORTORCOM + 0),100
VPOKE (PORTORCOM + 1),50
VPOKE (PORTORCOM + 2),0
VPOKE (PORTORCOM + 3),10
VPOKE (FACHACOM + 0),255
VPOKE (FACHACOM + 1),129
VPOKE (FACHACOM + 2),129
VPOKE (FACHACOM + 3),129
VPOKE (FACHACOM + 4),129
VPOKE (FACHACOM + 5),129
VPOKE (FACHACOM + 6),129
VPOKE (FACHACOM + 7),255
```

Ahora deberías tener en el medio de la pantalla una Figura de forma rectangular. Ahora teclea:

```
VDP(1) = 225
```

Verás que la figura se ha **duplicado** en tamaño. Esta es la segunda talla de Figuras. Cambia las coordenadas vertical y horizontal del 'Portor' para verla moverse. Ahora teclea VDP(1) = 224 para volver a Figuras pequeñas. La siguiente talla de Figura une cuatro cuadratines 8 * 8 puntos, como se muestra en el siguiente diagrama:



"Tetrafiguras"
"Cuadratines 16*16"

Para el 'Portor' que vaya a pasear esa Figura 'cuádruple' por la pantalla, esa es una única Figura, así que los valores que metas como coordenadas vertical y horizontal en la zona pertinente de la VRAM siguen correspondiendo al punto situado -como siempre- en la esquina superior izquierda.

El siguiente pequeño programa en BASIC utilizará VPOKE para **meter** los datos dentro de la VRAM para definir los 3/4 de Figura, que nos falta (pues aprovecharemos la anterior para que sea el 1/4 de arriba-derecha).

```
10 FOR A = 8 TO 31
20 READ B
30 VPOKE (FACHACOM + A),B
40 NEXT A
50 STOP
60 DATA 255,195,165,153,153,165,195,255
70 DATA 255,153,153,255,255,153,153,255
80 DATA 231,165,255,36,36,255,165,231
```

Ahora teclea VDP(1) = 226. Ese valor en ese **registro** interno hará que el sistema acepte las figuras de esa talla cuádruple. Intenta alterar los valores de las coordenadas vertical y horizontal de la figura para ver cómo se desplaza. Insisto en que en este modo, no son cuatro Figuras que estén unidas; está considerada y tratada como una sola Figura. Obviamente al ser de talla cuádruple las 256 Figuras posibles en talla 8 * 8 se quedan en 64 Figuras 16 * 16 posibles. Sigue habiendo, sin embargo, los mismos 32 **Portores** por lo que puedes 'llevar' a pantalla simultáneamente 32 de esas 64 Figuras.

La última tabla de Figuras es idéntica al del modo anterior, excepto que las figuras están duplicadas en tamaño al ser 'portadas' a pantalla. Este MODO 4 de FIGURAS se obtiene 'hincando' en el registro interno pertinente el valor 227. Es decir:

LET VDP(1) = 227

VRAM=Memoria de Escritura/Lectura para Vídeo.

La VRAM contiene 16384 celdillas de un octeto que pueden estar distribuidas de varias maneras diferentes. Cuando escribes un juego, es tu decisión cómo repartir la VRAM dentro de ciertas restricciones. El circuito procesador que controla la salida de vídeo para obtener las imágenes tiene 8 registros, numerados del 0 al 7, cuyo nombre clave es VDP(mid), siendo 'mid' el Número IDentificativo del registro y VDP el 'acrónimo' (i.e. las siglas) de ViDeo Procesador. El valor registrado en cada VDP determina cómo será distribuida la VRAM.Si te dedicas a dar valores aleatoriamente (i.e. a lo loco) podría provocar que al ordenador se le 'gripen las meninges' y tendrás que apagarlo o restaurarlo (i.e. volver a instaurar las condiciones iniciales). Por tanto, puedes registrar en los VDPs los valores que se te antojen aunque no hayan sido cuidadosamente calculados y comprobados. Eso no los estropea, pero tampoco conseguirás lo que buscas... de una forma metódica.

El primer VDP, VDP(0), no es de ningún interés para nosotros y te aconsejamos que no alteres su valor. (Aunque allá tú).

El principal uso del VDP(1) es controlar la talla y la duplicación de talla de la figura. Si a VDP(1) se le da un valor de 224, cada una de las 256 Figuras -de las que sólo 32 son simultáneamente portables a pantalla- necesitará 8 octetos para definir su forma o tipo. Cada figura aparecerá en la pantalla como un carácter de tamaño normal, i.e. ocupando un cuadratín. Cuando a VDP(1) se le da un valor de 255 sigue con los 8 octetos que le dan su forma, pero la mota proyectada en pantalla es de dos veces el tamaño del modo anterior.

Si VDP(1) tiene un valor de 226, entonces se usan 32 octetos para definir la forma o tipo de la figura; y aparecerá en pantalla ocupando cuatro cuadratines de tamaño normal contiguos para hacer un cuadrado de 16*16 puntos. Cuando VDP(1) tiene un valor de 227 es similar a cuando es 226; por lo que las figuras son de 16*16 bits pero al proyectarse en pantalla se amplía al doble el tamaño de la mota correspondiente. El valor de VDP(2) determina la dirección donde comienza el mapa de pantalla, o 'pizarra' interna en estrecha correlación con la gradilla de 24 filas por 32 columnas de la pantalla.

Hay 16 posiciones posibles para este mapa de pantalla, como se muestra en la siguiente tabla:

VDP 2	Dirección de comienzo del mapa de pantalla
0	0
1	1024
2	2048
3	3072
4	4096
5	5120
6	6144
7	7168
8	8192
9	9216
10	10240
11	11264
12	12288
13	13312
14	14336
15	15360

El valor de VDP(3) determina la dirección donde comienza la tabla de colores. VDP(3) puede ponerse a cualquier valor entre 0 y 255. La tabla de color empezará en la dirección obtenida al multiplicar el valor de VDP(3) por 64. Así, si VDP(3) tiene un valor de 21, la tabla COLOR comenzará en 21*64 = 1344.

El valor de VDP(4) determina la dirección donde comienza la tabla de TIPOS donde se refleja la forma de los 256 caracteres del repertorio. Sólo puede asignársele una de las siguientes ocho posibilidades:

VDP 4	Dirección de comiento del tipo de caracteres
0	0
1	2048
2	4096
3	6144
4	8192
5	10240
6	12288
7	14336

El valor de VDP(5) determina la dirección donde comienza la tabla de atributos o de Portores de las Figuras. Su valor puede ser cualquiera entre 0 y 127 inclusive. Esta dirección de comienzo se calcula entonces multiplicando el valor de VDP(5) por 128. Así, si VDP(5) tiene un valor de 45, la tabla de Portores comenzará en la dirección 45*128 = 5760 de la VRAM.

El valor de VDP(6) determina la dirección donde comienza la tabla con las formas o Tipos de las Figuras. Puede ser cualquiera de los ocho valores mostrados en la siguiente tabla:

VDP 6	Dirección de comienzo del tipo de figuras
0	0
1	2048
2	4096
3	6144
4	8192
5	10240
6	12288
7	14336

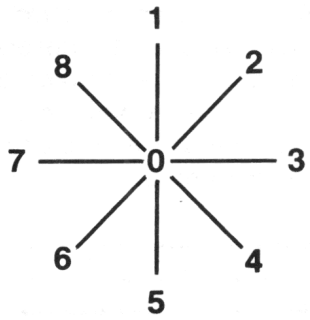
VDP(7) no es de mucho interés para el programador en Código Máquina. Su valor determinará qué color de pluma y papel se aplicará en el Modo de pantalla 0 cuando se escriban textos.

MANDOS O 'CONTROLES MANUALES' (JOY-STICKS)

Todos los aparatos MSX tienen la facilidad de **acoplar** uno o dos mandos. Hay dos rutinas en la memoria ROM que pueden usarse para determinar la condición de cada mando. La primera rutina inspecciona el circuito de entrada de mando especificado cuando se trata de un mando de **RUMBO.**

Para usar esta rutina debes cargar primero el registro A con 1 ó 2 -de acuerdo con el mando que deseas leer- luego citar la rutina que está en la celdilla **213.** La rutina inspeccionará la posición del mando, y pondrá un valor en el registro A que corresponda a esa posición. El siguiente diagrama muestra los valores que representan las ocho direcciones de un mando de 'rumbo'.

Si el mando está en la posición central, el registro A contendrá 0:



Aquí está una pequeña rutina en Código Máquina para demostrar cómo se efectúa la **lectura** de los mandos. Primero se carga el registro A con 1 (el número del mando que vamos a inspeccionar). Luego se **cita** ('CD' en Código Máquina) la rutina que comienza en la dirección 005D (la 213 hablando en decimal). El valor del registro A se coloca luego en la celdilla 50000 (i.e. C350 en **dé**cimohe**xal**):

Dirección de comienzo	40000
Dirección de final	40008
Suma total de los octetos	1007

```
3E01
CDD500
3250C3
C9
```

El siguiente programa en BASIC puede usarse para comprobar la rutina anterior:

```
1000 SCREEN 1
1005  DEF USR = 40000
1010 A = USR(0)
1015 B = PEEK (50000)
1020 PRINT B
1025 GOTO 1010
```

Ahora manda que ejecute el programa anterior. En pantalla **expondrá** el valor que corresponde a la posición del mando uno.

La segunda rutina asociada con los mandos se usa para ver si está pulsado o no el **botón** de disparo de los mandos de rumbo. La rutina está a partir de la dirección 216 y, como con la rutina anterior, el registro A debe ser cargado con 1 ó 2 de acuerdo con el mando cuyo botón queremos inspeccionar.

La rutina entonces comprobará el botón de disparo pertinente; si está pulsado cargará el registro A con 255, y si no está pulsado lo cargará con 0. La siguiente rutina en Código Máquina demostrará el uso de esta rutina de detección del botón de disparo. (No la teclees a menos que tengas el mando, porque la única forma de salir de la rutina y volver a BASIC es pulsando el botón de **restauro** del sistema. Pero, claro que allá tú... yo no lo impido; sino que más bien lo pido.

Dirección de comienzo	**40000**
Dirección de final	**40008**
Suma total de los octetos	**1007**

```
3E01
CDD800
3C
20F8
C9
```

Mete y ejecuta el siguiente programa en BASIC para comprobar la rutina anterior:

```
1000 DEF USR = 40000
1005 A = USR(0)
1010 CLS
1015 PRINT "HA SIDO PULSADO EL BOTON
     DE DISPARO"
```

SISTEMA BASICO DE ENTRADA/SALIDA (BIOS)

El BIOS fue brevemente descrito en el capítulo concerniente al programa Invasor Espacial. El BIOS contiene muchas rutinas en Código Máquina que podemos aprovechar; con lo que nos será más fácil escribir una gran cantidad de programas en Código Máquina. Las **rutinas** que he listado aquí son las únicas que considero importantes durante el aprendizaje de la programación en Código Máquina.

Nombre: Llenar la VRAM con un dato.

Función: Cuando se apela a esta rutina se llenará el área especificada de la VRAM con el octeto mencionado. Por ejemplo, sería usada para llenar los 768 bytes del mapa de pantalla en el modo de pantalla 1 con el carácter 32 -que es el blanco- lo cual tendría el efecto de limpiar la pantalla, i.e. dejaría en 'claro'.

Requisitos: La dirección de comienzo del bloque de la VRAM que ha de llenarse, debe estar contenida en el registro HL. La longitud del bloque debe registrarse en BC y el octeto, el del valor 32 en el ejemplo anterior, en el registro A.

CODOPE: CD 56 00='acude' a la dirección 0056h (86 en dec).

Nombre: Mover un bloque de la VRAM hasta la memoria RAM de usuario.

Función: Esta rutina moverá el bloque especificado de la VRAM hasta la memoria ordinaria del usuario. Podría usarse para copiar el 'mapa' de la VRAM de la imagen en pantalla sobre cualquier zona de la Memoria de Escritura/Lectura.

Requisitos: El registro HL debe contener la dirección de comienzo en la VRAM y el registro DE la dirección de comienzo en la RAM a partir de la cual se depositará la copia, y el registro BC debe contener la longitud del bloque que se mueve.

CODOPE: CD 59 00='acude' a la dirección 0059h (89 en dec).

Nombre: Mover un bloque de memoria hasta la VRAM.

Función: Esta rutina se usa para copiar el bloque de memoria de Escritura/Lectura especificado, sobre la memoria de vídeo. Podría usarse para copiar un repertorio de caracteres con nuevos tipos diseñados por el usuario y almacenados en algún lugar de la RAM, sobre las direcciones que normalmente ocupan en la VRAM, en los modos de texto.

Requisitos: El registro HL debe contener la dirección de comienzo del bloque de memoria en la RAM que va a moverse. El registro DE debe contener la dirección de comienzo en la VRAM a partir de la cual se depositará la copia. El registro BC debe contener la longitud del bloque que interviene en el movimiento.

CODOPE: CD 5C 00='acuda' a la dirección 005Ch (92 en dec).

Nombre: Recoger el carácter INpuesto por teclado.

Función: Cuando se apela a esta rutina, esperará hasta que se pulse cualquier tecla y luego ingresa en el registro A el código del carácter tecleado. Un ejemplo típico es cuando quieres que alguien pulse una sola letra o cifra para elegir en un menú de opciones.

CODOPE: CD 9F 00='acuda' a la dirección 009F (159 en dec).

Nombre: Situar el cursor en una posición.

Función: Esta rutina colocará el cursor en el sitio de la pantalla especificado. Podría aprovecharse para señalar dónde una persona debería empezar a contestar una pregunta que has expuesto en pantalla.

Requisitos: El registro H debe contener el número de columna en la que deseas colocar el cursor. En el modo de pantalla 1 debiera estar en la banda de 0 a 31. El registro L debe contener el número de fila en la que deseas colocar el cursor. E en el modo 1 debiera estar en la banda de 0 a 23.

CODOPE: CD C6 00='acuda' a la dirección 00C6h (198 en dec).

Nombre: Oculta los literales adscritos a las teclas Funcionales.

Función: Cuando se recurre a esta rutina hacemos que desaparezcan de la última fila de la pantalla dichos literales.

CODOPE: CD CC 00='acuda' a la dirección 00CCh (204 en dec).

Nombre: Exhibe los literales asignados a las teclas Funcionales.

Función: Cuando se cita esta rutina, facultamos sean exhibidos en la última línea de la imagen dichos literales.

CODOPE: CD CF 00='acuda' a la dirección 00CFh (207 en dec).

Nombre: Meter un valor en una celdilla de la VRAM.

Función: Esta rutina es la equivalente en Código Máquina de la instrucción en BASIC VPOKE. La dirección de la celdilla en la VRAM es una dirección entre 0 y 16383. (La VRAM tiene 16K de capacidad) y el dato que se mete es un octeto, i.e. un número entre 0 y 255.

Requisitos: El registro HL debe contener la dirección en la VRAM donde quieres escribir. El registro A debe contener el número entre 0 y 255 que escribes.

CODOPE: CD 4D 00='acuda' a la dirección 004Dh (77 en dec).

Nombre: Mirar lo que hay en una celdilla de la VRAM.

Función: Esta rutina es la equivalente en Código Máquina de la instrucción en BASIC VPEEK. Después de ejecutar esta rutina el registro A contendrá el valor contenido en la celdilla de VRAM especificada.

Requisitos: El registro HL debe contener la dirección de la celdilla en la VRAM donde deseas examinar lo que hay.

CODOPE: CD 4A 00='acuda' a la dirección 004Ah (74 en dec).

Algunas Rutinas Aprovechables

Algunas Rutinas Aprovechables

Este capítulo contiene unas cuantas rutinas útiles para que las teclees. Demuestran efectivamente el poder de la programación en Código Máquina. Cada rutina está alojada a partir de una dirección diferente en la memoria; por lo tanto es posble, si lo deseas, tener más de una rutina en la memoria al mismo tiempo. Está disponible el libro titulado "Useful Utilities for Your MSX" de la editorial Virgin Book. Contiene muchos utensilios de programación útiles que puedes usar si no tienes ganas de escribirlas tú.

Nombre: "Desrrollar" una línea de texto a la derecha.

Función: Esta rutina te permitirá desplazar cualquiera de los 24 renglones de texto en el modo 1 de pantalla una posición a la derecha.

Requisitos: La celdilla de memoria 50000 debe contener un número entre 0 y 23 representando el número de fila identificador de la que deseas correr a la derecha.

Dirección de comienzo 40000
Dirección de final 40044
Suma total de los octetos 4158

```
3A50C3    20          LD      A,(50000)
012000    30          LD      BC,32
211F18    40          LD      HL,6175
3C        50          INC     A
3D        60 LOOP:    DEC     A
CA529C    70          JP      Z,LOOP1
09        80          ADD     HL,BC
C34A9C    90          JP      LOOP
CD4A00   100 LOOP1:   CALL    74
F5       110          PUSH    AF
011F00   120          LD      BC,31
2B       130 LOOP2:   DEC     HL
CD4A00   140          CALL    74
23       150          INC     HL
CD4D00   160          CALL    77
2B       170          DEC     HL
0B       180          DEC     BC
79       190          LD      A,C
B0       200          OR      B
C2599C   210          JP      NZ,LOOP2
F1       220          POP     AF
CD4D00   230          CALL    77
C9       240          RET
```

Nombre: "Desrrollar" una línea de texto a la izquierda.

Función: Esta rutina te permitirá desplazar cualquiera de las 24 filas del texto en el modo 1 de pantalla una posición a la izquierda.

Requisitos: La casilla de memoria debe contener un número entre 0 y 23 representando la fila que deseas correr.

Dirección de comienzo	**40100**
Dirección de final	**40144**
Suma total de los octetos	**4420**

```
3A51C3    20          LD      A,(50001)
012000    30          LD      BC,32
210018    40          LD      HL,6144
3C        50          INC     A
3D        60 LOOP:    DEC     A
CAB69C    70          JP      Z,LOOP1
09        80          ADD     HL,BC
C3AE9C    90          JP      LOOP
CD4A00    100 LOOP1:  CALL    74
F5        110         PUSH    AF
011F00    120         LD      BC,31
23        130 LOOP2:  INC     HL
CD4A00    140         CALL    74
2B        150         DEC     HL
CD4D00    160         CALL    77
23        170         INC     HL
0B        180         DEC     BC
79        190         LD      A,C
B0        200         OR      B
C2BD9C    210         JP      NZ,LOOP2
F1        220         POP     AF
CD4D00    230         CALL    77
C9        240         RET
```

Nombre: "Desrrollar" una columna de texto en el modo 1 de pantalla una posición hacia arriba.

Función: Esta rutina te permitirá desplazar cualquiera de las 32 columnas de texto en el modo 1 de pantalla una posición hacia arriba.

Requisitos: La celdilla de memoria 50002 debe contener un número entre 0 y 31 que es el número de columna identificador de la que deseas subir.

Dirección de comienzo 40300
Dirección de final 40345
Suma total de los octetos 4540

```
3A52C3    20          LD      A,(50002)
3C        30          INC     A
210018    40          LD      HL,6144
3D        50 LOOP:    DEC     A
CA7B9D    60          JP      Z,LOOP1
23        70          INC     HL
C3739D    80          JP      LOOP
CD4A00    90 LOOP1:   CALL    74
F5       100          PUSH    AF
011700   110          LD      BC,23
112000   120          LD      DE,32
19       130 LOOP2:   ADD     HL,DE
CD4A00   140          CALL    74
ED52     150          SBC     HL,DE
CD4D00   160          CALL    77
19       170          ADD     HL,DE
0B       180          DEC     BC
79       190          LD      A,C
B0       200          OR      B
C2859D   210          JP      NZ,LOOP2
F1       220          POP     AF
CD4D00   225          CALL    77
C9       230          RET
```

Nombre: "Desrrollar una columna de texto hacia abajo.
Función: Esta rutina te permitirá desplazar cualquiera de las 32 columnas de texto en el modo 1 de pantalla una posición hacia abajo.
Requisitos: La celdilla de memoria 50003 debe contener un número entre 0 y 31 representando la columna que deseas bajar.

Dirección de comienzo 40200
Dirección de final 40246
Suma total de los octetos 4761

```
3A53C3    20          LD      A,(50003)
3C        30          INC     A
21E01A    40          LD      HL,6880
3D        50 LOOP:    DEC     A
CA179D    60          JP      Z,LOOP1
23        70          INC     HL
```

```
C30F9D      80          JP      LOOP
CD4A00      90 LOOP1:   CALL    74
F5         100          PUSH    AF
011700     110          LD      BC,23
112000     120          LD      DE,32
ED52       130 LOOP2:   SBC     HL,DE
CD4A00     140          CALL    74
19         150          ADD     HL,DE
CD4D00     160          CALL    77
ED52       170          SBC     HL,DE
0B         180          DEC     BC
79         190          LD      A,C
B0         200          OR      B
C2219D     210          JP      NZ,LOOP2
F1         220          POP     AF
CD4D00     230          CALL    77
C9         240          RET
```

```
1000 SCREEN 1
1005 VDP(2)=6
1010 X=65
1015 FOR A=6144 TO 6880 STEP 32
1020 Y=X
1025 FOR B=0 TO 31
1030 VPOKE (A+B),Y
1035 Y=Y+1
1040 NEXT B
1045 X=X+1
1050 NEXT A
1055 POKE (50000),10
1060 DEF USR=40000
1065 A=USR(1)
1070 GOTO 1065
```

El programa en BASIC se puede usar para commprobar cualquiera de las cuatro rutinas de 'desrrolle'. Necesitarás alterar las líneas 1055 y 1060 dependiendo de la rutina que estés comprobando y de la columna o fila que desees correr.

Nombre: Efecto sonoro de disparo laser.
Función: Cuando se cite esta rutina producirá un sonido que es el típico sonido asociado con las máquinas de juegos de guerreros y galácticos.

Dirección de comienzo 40600
Dirección de final 40667
Suma total de los octetos 7249

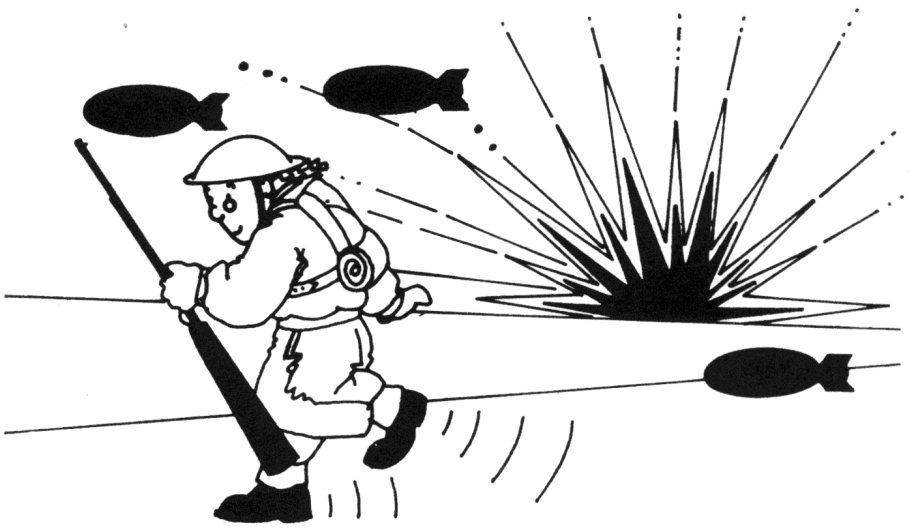
```
3E08    20          LD      A,8
1E0F    30          LD      E,15
CD9300  40          CALL    147
3E07    50          LD      A,7
1EFE    60          LD      E,254
CD9300  70          CALL    147
3E00    80          LD      A,0
1E6E    90          LD      E,110
CD9300  100         CALL    147
3E01    110         LD      A,1
1E00    120         LD      E,0
CD9300  130         CALL    147
1E6E    140         LD      E,110
3E00    150 LOOP:   LD      A,0
CD9300  160         CALL    147
3EC8    170         LD      A,200
F5      180 DELAY:  PUSH    AF
3E0A    190         LD      A,10
3D      200 DEL:    DEC     A
C2C09E  210         JP      NZ,DEL
F1      220         POP     AF
3D      230         DEC     A
C2BD9E  240         JP      NZ,DELAY
7B      250         LD      A,E
C606    260         ADD     A,6
CAD49E  270         JP      Z,END
3D      280         DEC     A
5F      290         LD      E,A
C3B69E  300         JP      LOOP
3E07    310 END:    LD      A,7
1EFF    320         LD      E,255
CD9300  330         CALL    147
C9      340         RET
```

Nombre: Efecto sonoro de una bomba.
Función: Esta rutina producirá el sibilante sonido que hace una bomba al caer y luego el efecto de la explosión.

Dirección de comienzo	40400
Dirección de final	40514
Suma total de los octetos	13909

```
3E07    20          LD      A,7
1EFE    30          LD      E,254
CD9300  40          CALL    147
3E0A    50          LD      A,8
1E0F    60          LD      E,15
CD9300  70          CALL    147
1E28    80          LD      E,40
3E00    90 LOOP:    LD      A,0
CD9300  100         CALL    147
3E0A    110         LD      A,10
F5      120 DELAY:  PUSH    AF
3EFF    130         LD      A,255
3D      140 DEL:    DEC     A
C2EA9D  150         JP      NZ,DEL
F1      160         POP     AF
3D      170         DEC     A
C2E79D  180         JP      NZ,DELAY
7B      190         LD      A,E
D696    200         SUB     150
CAFF9D  210         JP      Z,NEXT
C697    220         ADD     A,151
5F      230         LD      E,A
C3E09D  240         JP      LOOP
3E00    250 NEXT:   LD      A,0
1E00    260         LD      E,0
CD9300  270         CALL    147
3E07    280         LD      A,7
1EF7    290         LD      E,247
CD9300  300         CALL    147
3E00    310         LD      A,0
F5      320 NEXT1:  PUSH    AF
5F      330         LD      E,A
CD9300  340         CALL    147
3E32    350         LD      A,50
F5      360 STEVE:  PUSH    AF
3EFF    370         LD      A,255
3D      380 TRACEY: DEC     A
C2199E  390         JP      NZ,TRACEY
F1      400         POP     AF
```

```
3D          410             DEG     A
C2169E      420             JP      NZ,STEVE
F1          430             POP     AF
D61F        440             SUB     31
CA2D9E      450             JP      Z,LDEL
C620        460             ADD     A,32
C30F9E      470             JP      NEXT1
3E64        480 LDEL:       LD      A,100
F5          490 LDEL1:      PUSH    AF
3EFF        500             LD      A,255
3D          510 LDEL2:      DEC     A
C2329E      520             JP      NZ,LDEL2
F1          530             POP     AF
3D          540             DEC     A
C22F9E      550             JP      NZ,LDEL1
3E07        560             LD      A,7
1EFF        570             LD      E,255
CD9300      580             CALL    147
C9          590             RET
```



Apéndices

Apéndices

1) CODigos de OPEración del Z80
2) Conversión de decimal a base 16
3) Binarios
4) Diseñador de caracteres/figuras

Codopes

ADC A, (HL)	8E
ADC A, (IX + d)	DD8Ed
ADC A, (IY + d)	FD8Ed
ADC A, A	8F
ADC A, B	88
ADC A, C	89
ADC A, D	8A
ADC A, E	8B
ADC A, H	8C
ADC A, L	8D
ADC A, n	CEn
ADC HL, BC	ED4A
ADC HL, DE	ED5A
ADC HL, HL	ED6A
ADC HL, SP	ED7A
ADD A, (HL)	86
ADD A, (IX + d)	DD86d
ADD A, (IY + d)	FD86d
ADD A, A	87
ADD A, B	80
ADD A, C	81
ADD A, D	82
ADD A, E	83
ADD A, H	84
ADD A, L	85
ADD A, n	C6n
ADD HL, BC	09
ADD HL, DE	19
ADD HL, HL	29
ADD HL, SP	39
ADD IX, BC	DD09
ADD IX, DE	DD19
ADD IX, IX	DD29
ADD IX, SP	DD39
ADD IY, BC	FD09
ADD IY, DE	FD19
ADD IY, IY	FD29
ADD IY, SP	FD39
AND(HL)	A6
AND (IX + d)	DDA6d
AND (IY + d)	FDA6d
AND A	A7
AND B	A0
AND C	A1
AND D	A2
AND E	A3
AND H	A4
AND L	A5
AND n	E6n
BIT 0, (HL)	CB46
BIT 0, (IX + D)	DDCBd46
BIT 0, (IY + d)	FDCBd46
BIT 0, A	CB47
BIT 0, B	CB40
BIT 0, C	CB41
BIT 0, D	CB42
BIT 0, E	CB43
BIT 0, H	CB44
BIT 0, L	CB45
BIT 1, (HL)	CB4E
BIT 1, (IX + d)	DDCBd4E
BIT 1, (IY + d)	FDCBd4E
BIT 1, A	CB4F
BIT 1, B	CB48
BIT 1, C	CB49
BIT 1, D	CB4A
BIT 1, E	CB4B
BIT 1, H	CB4C
BIT 1, L	CB4D
BIT 2, (HL)	CB56
BIT 2, (IX + d)	DDCBd56
BIT 2, (IY + d)	FDCBd56
BIT 2, A	CB57
BIT 2, B	CB50
BIT 2, C	CB51
BIT 2, D	CB52
BIT 2, E	CB53
BIT 2, H	CB54
BIT 2, L	CB55
BIT 3, (HL)	CB5E
BIT 3, (IX + d)	DDCBd5E
BIT 3, (IY + d)	FDCBd5E
BIT 3, A	CB5F
BIT 3, B	CB58
BIT 3, C	CB59
BIT 3, D	CB5A
BIT 3, E	CB5B
BIT 3, H	CB5C
BIT 3, L	CB5D
BIT 4, (HL)	CB66
BIT 4, (IX + d)	DDCBd66
BIT 4, (IY + d)	FDCBd66
BIT 4, A	CB67
BIT 4, B	CB60
BIT 4, C	CB61
BIT 4, D	CB62
BIT 4, E	CB63
BIT 4, H	CB64
BIT 4, L	CB65
BIT 5, (HL)	CB6E
BIT 5, (IX + d)	DDCBd6E
BIT 5, (IY + D)	FDCBd6E
BIT 5, A	CB6F
BIT 5, B	CB68
BIT 5, C	CB69
BIT 5, D	CB6A
BIT 5, E	CB6B
BIT 5, H	CB6C

BIT 5, L	CB6D
BIT 6, (HL)	CB76
BIT 6, (IX + d)	DDCBd76
BIT 6, (IY + d)	FDCBd76
BIT 6, A	C877
BIT 6, B	CB70
BIT 6, C	CB71
BIT 6, D	CB72
BIT 6, E	CB73
BIT 6, H	CB74
BIT 6, L	CB75
BIT 7, (HL)	CB7E
BIT 7, (IX + d)	DDCBd7E
BIT 7, (IY + d)	FDCBd7E
BIT 7, A	CB7F
BIT 7, B	CB78
BIT 7, C	CB79
BIT 7, D	CB7A
BIT 7, E	CB7B
BIT 7, H	CB7C
BIT 7, L	CB7D
CALL C, nn	DCnn
CALL M, nn	FCnn
CALL NC, nn	D4nn
CALL nn	CDnn
CALL NZ, nn	C4nn
CALL P, nn	F4nn
CALL PE, nn	ECnn
CALL PO, nn	E4nn
CALL Z, nn	CCnn
CCF	3F
CP (HL)	BE
CP (IX + d)	DDBEd
CP (IY + d)	FDBEd
CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD
CP n	FEn
CPD	EDA9
CPDR	EDB9
CPI	EDA1
CPIR	EDB1
CPL	2F
DAA	27
DEC (HL)	35
DEC (IX + d)	DD35d
DEC (IY + d)	FD35d
DEC A	3D
DEC B	05
DEC BC	0B
DEC C	0D
DEC D	15
DEC DE	1B
DEC E	1D
DEC H	25
DEC HL	2B
DEC IX	DD2B
DEC IY	FD2B
DEC L	2D
DEC SP	3B
DI	F3
DJNZ, d	10d
E1	FB
EX (SP), HL	E3
EX (SP), IX	DDE3
EX (SP), IY	FDE3
EX AF, AF	08
EX DE, HL	EB
EXX	D9
HALT	76
IM 0	ED46
IM 1	ED56
IM 2	ED5E
IN A, (C)	ED78
IN A, (n)	DBn
IN B, (C)	ED40
IN C, (C)	ED48
IN D, (C)	ED50
IN E, (C)	ED58
IN H, (C)	ED60
IN L, (C)	ED68
INC (HL)	34
INC (IX + d)	DD34d
INC (IY + d)	FD34d
INC A	3C
INC B	04
INC BC	03
INC C	0C
INC D	14
INC DE	13
INC E	1C
INC H	24
INC HL	23
INC IX	DD23
INC IY	FD23
INC L	2C
INC SP	33
IND	EDAA
INDR	EDBA
INI	EDA2
INIR	EDB2
JP (HL)	E9
JP (IX)	DDE9
JP (IY)	FDE9
JP C, nn	DAnn
JP M, nn	FAnn
JP NC, nn	D2nn
JP nn	C3nn

JP NZ, nn	C2nn
JP P, nn	F2nn
JP PE, nn	EAnn
JP PO, nn	E2nn
JP Z, nn	CAnn
JR C, d	38d
JR, d	18d
JR NC, d	30d
JR NZ, d	20d
JR Z, d	28d
LD (BC), A	02
LD (DE), A	12
LD (HL), A	77
LD (HL), B	70
LD (HL), C	71
LD (HL), D	72
LD (HL), E	73
LD (HL), H	74
LD (HL), L	75
LD (HL), n	36n
LD (IX + d), A	DD77d
LD (IX + d), B	DD70d
LD (IX + d), C	DD71d
LD (IX + d), D	DD72d
LD (IX + d), E	DD73d
LD (IX + d), H	DD74d
LD (IX + d), L	DD75d
LD (IX + d), n	DD36dn
LD (IY + d), A	FD77d
LD (IY + d), B	FD70d
LD (IY + d), C	FD71d
LD (IY + d), D	FD72d
LD (IY + d), E	FD73d
LD (IY + d), H	FD74d
LD (IY + d), L	FD75d
LD (IY + d), n	FD36dn
LD (nn), A	32nn
LD (nn), BC	ED43nn
LD (nn), DE	ED53nn
LD (nn), HL	22nn
LD (nn), IX	DD22nn
LD (nn), IY	FD22nn
LD (nn), SP	ED73nn
LD A, (BC)	0A
LD A, (DE)	1A
LD A, (HL)	7E
LD A, (IX + d)	DD7Ed
LD A, (IY + d)	FD7Ed
LD A, (nn)	3Ann
LD A, A	7F
LD A, B	78
LD A, C	79
LD A, D	7A
LD A, E	7B
LD A, H	7C
LD A, I	ED57
LD A, L	7D
LD A, n	3En
LD B, (HL)	46
LD B, (IX + d)	DD46d
LD B, (IY + d)	FD46d
LD B, A	47
LD B, B	40
LD B, C	41
LD B, D	42
LD B, E	43
LD B, H	44
LD B, L	45
LD B, n	06n
LD B, C (nn)	ED4Bnn
LD B, C nn	01nn
LD C, (HL)	4E
LD C, (IX + d)	DD4Ed
LD C, (IY + d)	FD4Ed
LD C, A	4F
LD C, B	48
LD C, C	49
LD C, D	4A
LD C, E	4B
LD C, H	4C
LD C, L	4D
LD C, n	0En
LD D, (HL)	56
LD D, (IX + d)	DD56d
LD D, (IY + d)	FD56d
LD D, A	57
LD D, B	50
LD D, C	51
LD D, D	52
LD D, E	53
LD D, H	54
LD D, L	55
LD D, n	16n
LD DE, (nn)	ED58nn
LD DE, nn	11nn
LD E, (HL)	5E
LD E, (IX + d)	DD5Ed
LD E, (IY + d)	FD5Ed
LD E, A	5F
LD E, B	58
LD E, C	59
LD E, D	5A
LD E, E	5B
LD E, H	5C
LD E, L	5D
LD E, n	1En
LD H, (HL)	66
LD H, (IX + d)	DD66d
LD H, (IY + d)	FF66d
LD H, A	67
LD H, B	60
LD H, C	61

LD H, D	62
LD H, E	63
LD H, H	64
LD H, L	65
LD H, n	26n
LD HL, (nn)	2Ann
LD HL, nn	21nn
LD I, A	ED47
LD IX, (nn)	DD2Ann
LD IX, nn	DD21nn
LD IY, (nn)	FD2Ann
LD IY, nn	FD21nn
LD L, (HL)	6E
LD L, (IX + d)	DD6Ed
LD L, (IY + d)	FD6Ed
LD L, A	6F
LD L, B	68
LD L, C	69
LD L, D	6A
LD L, E	6B
LD L, H	6C
LD L, L	6D
LD L, n	2En
LD SP, (nn)	ED7Bnn
LD SP, HL	F9
LD SP, IX	DDF9
LD SP, IY	FDF9
LD SP, nn	31nn
LDD	EDA8
LDDR	EDB8
LDI	EDAO
LDIR	EDB0
NEG	ED44
NOP	00
OR (HL)	B6
OR (IX + d)	DDB6d
OR (IY + d)	FDB6d
OR A	B7
OR B	B0
OR C	B1
OR D	B2
OR E	B3
OR H	B4
OR L	B5
OR n	F6n
OTDR	EDBB
OTIR	EDB3
OUT (C), A	ED79
OUT (C), B	ED41
OUT (C), C	ED49
OUT (C), D	ED51
OUT (C), E	ED59
OUT (C), H	ED61
OUT (C), L	ED69
OUT (n), A	D3n
OUTD	EDAB
OUTI	EDA3
POPAF	F1
POP BC	C1
POP DE	D1
POP HL	E1
POP IX	DDE1
POP IY	FDE1
PUSH AF	F5
PUSH BC	C5
PUSH DE	D5
PUSH HL	E5
PUSH IX	DDE5
PUSH IY	FDE5
RES 0, (HL)	CB86
RES 0, (IX + d)	DDCBd86
RES 0, (IY + d)	FDCBd86
RES 0, A	CB87
RES 0, B	CB80
RES 0, C	CB81
RES 0, D	CB82
RES 0, E	CB83
RES 0, H	CB84
RES 0, L	CB85
RES 1, (HL)	CB8E
RES 1, (IX + d)	DDCBd8E
RES 1, (IY + d)	FDCBd8E
RES 1, A	CB8F
RES 1, B	CB88
RES 1, C	CB89
RES 1, D	CB8A
RES 1, E	CB8B
RES 1, H	CB8C
RES 1, L	CB8D
RES 2, (HL)	CB96
RES 2, (IX + d)	DDCBd96
RES 2, (IY + d)	FDCBd96
RES 2, A	CB97
RES 2, B	CB90
RES 2, C	CB91
RES 2, D	CB92
RES 2, E	CB93
RES 2, H	CB94
RES 2, L	CB95
RES 3, (HL)	CB9E
RES 3, (IX + d)	DDCBd9E
RES 3, (IY + d)	FDCBd9E
RES 3, A	CB9F
RES 3, B	CB98
RES 3, C	CB99
RES 3, D	CB9A
RES 3, E	CB9B
RES 3, H	CB9C
RES 3, L	CB9D
RES 4, (HL)	CBA6
RES 4, (IX + d)	DDCBdA6
RES 4, (IY + d)	FDCBdA6

RES 4, A	CBA7
RES 4, B	CBA0
RES 4, C	CBA1
RES 4, D	CBA2
RES 4, E	CBA3
RES 4, H	CBA4
RES 4, L	CBA5
RES 5, (HL)	CBAE
RES 5, (IX + d)	DDCBdAE
RES 5, (IY + d)	FDCBdAE
RES 5, A	CBAF
RES 5, B	CBA8
RES 5, C	CBA9
RES 5, D	CBAA
RES 5, E	CBAB
RES 5, H	CBAC
RES 5, L	CBAD
RES 6, (HL)	CBB6
RES 6, (IX + d)	DDCBdB6
RES 6, (IY + d)	FDCBdB6
RES 6, A	CBB7
RES 6, B	CBB0
RES 6, C	CBB1
RES 6, D	CBB2
RES 6, E	CBB3
RES 6, H	CBB4
RES 6, L	CBB5
RES 7, (HL)	CBBE
RES 7, (IX + d)	DDCBdBE
RES 7, (IY + d)	FDCBdBE
RES 7, A	CBBF
RES 7, B	CBB8
RES 7, C	CBB9
RES 7, D	CBBA
RES 7, E	CBBB
RES 7, H	CBBC
RES 7, L	CBBD
RET	C9
RET C	D8
RET M	F8
RET NC	D0
RET NZ	C0
RET P	F0
RET PE	E8
RET P0	E0
RET Z	C8
RETI	ED4D
RETN	ED45
RL (HL)	CB16
RL (IX + d)	DDCBd16
RL (IY + d)	FDCBd16
RL A	CB17
RL B	CB10
RL C	CB11
RL D	CB12
RL E	CB13
RL H	CB14
RL L	CB15
RLA	17
RLC (HL)	CB06
RLC (IX + d)	DDCBd06
RLC (IY + d)	FDCBd06
RLC A	CB07
RLC B	CB00
RLC C	CB01
RLC D	CB02
RLC E	CB03
RLC H	CB04
RLC L	CB05
RLCA	07
RLD	ED6F
RR (HL)	CB1E
RR (IX + d)	DDC8d1E
RR (IY + d)	FDCBd1E
RR A	CB1F
RR B	CB18
RR C	CB19
RR D	CB1A
RR E	CB1B
RR H	CB1C
RR L	CB1D
RRA	1F
RRC (HL)	CB0E
RRC (IX + d)	DDCBd0E
RRC (IY + d)	FDCBd0E
RRC A	CB0F
RRC B	CB08
RRC C	CB09
RRC D	CB0A
RRC E	CB0B
RRC H	CB0C
RRC L	CB0D
RRC A	0F
RRD	ED67
RST 0	C7
RST10H	D7
RST 18H	DF
RST 20H	E7
RST 28H	EF
RST 30H	F7
RST 38H	FF
RST 8	CF
SBC A, (HL)	9E
SBC A, (IX + d)	DD9Ed
SBC A, (IY + d)	FD9Ed
SBC A, A	9F
SBC A, B	98
SBC A, C	99
SBC A, D	9A
SBC A, E	9B
SBC A, H	9C
SBC A, L	9D

SBC A, n	DEn
SBC HL, BC	ED42
SBC HL, DE	ED52
SBC HL, HL	ED62
SBC HL, SP	ED72
SCF	37
SET 0, (HL)	CBC6
SET 0, (IX + d)	DDCBdC6
SET 0, (IY + d)	FDCBdC6
SET 0, A	CBC7
SET 0, B	CBC0
SET 0, C	CBC1
SET 0, D	CBC2
SET 0, E	CBC3
SET 0, H	CBC4
SET 0, L	CBC5
SET 1, (HL)	CBCE
SET 1, (IX + d)	DDCBdCE
SET 1, (IY + d)	FDCBdCE
SET 1, A	CBCF
SET 1, B	CBC8
SET 1, C	CBC9
SET 1, D	CBCA
SET 1, E	CBCB
SET 1, H	CBCC
SET 1, L	CBCD
SET 2, (HL)	CBD6
SET 2, (IX + d)	DDCBdD6
SET 2, (IY + d)	FDCBdD6
SET 2, A	CBD7
SET 2, B	CBD0
SET 2, C	CBD1
SET 2, D	CBD2
SET 2, E	CBD3
SET 2, H	CBD4
SET 2, L	CBD5
SET 3, (HL)	CBDE
SET 3, (IX + d)	DDCBdDE
SET 3, (IY + d)	FDCBdDE
SET 3, A	CBDF
SET 3, B	CBD8
SET 3, C	CBD9
SET 3, D	CBDA
SET 3, E	CBDB
SET 3, H	CBDC
SET 3, L	CBDD
SET 4, (HL)	CBE6
SET 4, (IX + d)	DDCbE6
SET 4, (IY + d)	FDCBdE6
SET 4, A	CBE7
SET 4, B	CBE0
SET 4, C	CBE1
SET 4, D	CBE2
SET 4, E	CBE2
SET 4, H	CBE4
SET 4, L	CBE5
SET 5, (HL)	CBEE
SET 5, (IX + d)	DDCBdEE
SET 5, (IY + d)	FDCBdEE
SET 5, A	CBEF
SET 5, B	CBE8
SET 5, C	CBE9
SET 5, D	CBEA
SET 5, E	CBEB
SET 5, H	CBEC
SET 5, L	CBED
SET 6, (HL)	CBF6
SET 6, (IX + d)	DDCBdF6
SET 6, (IY + d)	FDCBdF6
SET 6, A	CBF7
SET 6, B	CBF0
SET 6, C	CBF1
SET 6, D	CBF2
SET 6, E	CBF3
SET 6, H	CBF4
SET 6, L	CBF5
SET 7, (HL)	CBFE
SET 7, (IX + d)	DDCBdFE
SET 7, (IY + d)	FDCBdFE
SET 7, A	CBFF
SET 7, B	CBF8
SET 7, C	CBF9
SET 7, D	CBFA
SET 7, E	CBFB
SET 7, H	CBFC
SET 7, L	CBFD
SLA (HL)	CB26
SLA (IX + d)	DDCBd26
SLA (IY + d)	FDCBd26
SLA A	CB27
SLA B	CB20
SLA C	CB21
SLA D	CB22
SLA E	CB23
SLA H	CB24
SLA L	CB25
SRA (HL)	CB2E
SRA (IX + d)	DDCBd2E
SRA (IY + d)	FDCBd2E
SRA A	CB2F
SRA B	CB28
SRA C	CB29
SRA D	CB2A
SRA E	CB2B
SRA H	CB2C
SRA L	CB2D
SRL (HL)	CB3E
SRL (IX + d)	DDCBd3E
SRL (IY + d)	FDCBd3E
SRL A	CB3F
SRL B	CB38
SRL C	CB39

SRL D	CB3A
SRL E	CB3B
SRL H	CB3C
SRL L	CB3D
SUB (HL)	96
SUB (IX + d)	DD96d
SUB (IY + d)	FD96d
SUB A	97
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB L	95
SUB n	D6n
XOR (HL)	AE
XOR (IX + d)	DDAEd
XOR (IY + d)	FDAEd
XOR A	AF
XOR B	A8
XOR C	A9
XOR D	AA
XOR E	AB
XOR H	AC
XOR L	AD
XOR n	EEn

Conversión de Decimal a Base 16

	0	1	2	3	4	5	6	7	8	9	OA	OB	OC	OD	OE	OF
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Binarios

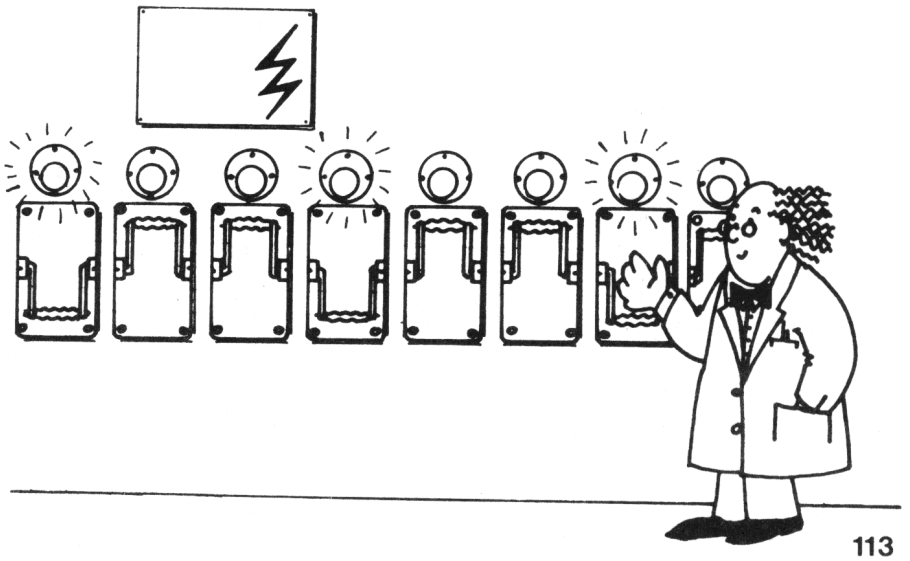
Si esta es la primera vez que te has encontrado con el mundo Binario, probablementee lo encontrarás ligeramente confuso. Incluso puede que te haya quitado las ganas de aprender Código Máquina, y sinceramente me gustaría que esto no ocurriese. Permíteme decirte que no necesitas entender el sistema Binario para escribir programas en Código Máquina. En la sección de almacenamiento de números, habrás aprendido que cada celdilla de memoria puede contener un número entre 0 y 255. Nos referimos a estos números como octetos; (ya que 256 = 2) y cada uno se compone de ocho elementos, conocidos como bits; y representados por uno de dos símbolos del sistema binario. Obviamente, en el habla inglesa no se llaman octetos, sino 'bytes' para reforzar el hecho que los los 'mordiscos' que el procesador da al tratar con la memoria. Para formar el octeto se ordenan y numeran así:

Ocho bits hacen un octeto

7	6	5	4	3	2	1	0

Estos ocho bits pueden ser considerados como si fueran ocho pequeños conmutadores. Cada conmutador puede estar apagado, bajado, abierto, encendido, alzado, cerrado, etc. Cuando un conmutador está en una posición, decimos que tiene un valor de 0. Cuando está en la otra posible, representa un valor de 1. Además su valor efectivo depende como en todo sistema de numeración del sitio u orden que ocupa en la retahíla, en la forma siguiente:

Peso del bit	128	64	32	16	8	4	2	1
Orden del bit	7	6	5	4	3	2	1	0



Puede verse pues que cuando el bit 4 está al valor 1 representa un valor efectivo de 16. Si los bits 1 y 3 estuviesen 'alzados' y el resto 'bajados', el octeto tendría un valor total de 10. Esto es porque el bit 1, cuando está al valor 1 tiene un peso de 2 y el bit 3 un peso de 8. Cualquier número entre 0 y 255 puede reflejarse como un octeto, como una combinación concreta de 8 bits. Afortunadamente, no tienes que interesarte en practicar esto: cuando usas POKE y PEEK -o el equivalente en Código Máquina- es el procesador el que se ocupa de los bits, y tú manejas números enteros en el sistema de base 10.

Si los bits 1, 4 y 6 de un octeto están al valor 1 ¿cuál es el octeto equivalente?

¿Qué bits de un octeto deberían estar al valor 1 para darle un valor de 67?

(No te he mostrado cómo calcular esto, pero mira a ver, porque estoy seguro que eres capaz de desarrollarlo por ti mismo).

Diseñador Carácter/Figura

El siguiente programa te permitirá diseñar la forma o tipo propio de cada carácter y/o Figura para usar en futuros programas. En un capítulo anterior te mostré cómo cada carácter estaba formado según ocho rayas moteadas horizontalmente y se describía mediante 8 octetos; este programa te permite dibujar esas 8 rayas y calcular los 8 octetos. Recuerda, este diseñador es para usar específicamente con el modo de pantalla 1. Introduce el programa usando METDEX y compruébalo de forma normal.

Dirección de comienzo	40000
Dirección de final	40702
Suma total de los octetos	85482

```
011800    110 START:  LD    BC,24
11F803    120         LD    DE,1016
215F9D    130         LD    HL,IXSTR
CD5C00    140         CALL  92
3E01      150         LD    A,1
32E3D6    160         LD    (CURCHR),A
32E8D6    170         LD    (VER),A
32E9D6    180         LD    (HOR),A
210218    190         LD    HL,6146
22E6D6    200         LD    (CURSOR),HL
21A08C    210         LD    HL,SETSTR
22E4D6    220         LD    (CHRADD),HL
3E80      230         LD    A,128
32E2D6    240         LD    (CHBIT),A
21D8D6    250         LD    HL,GRID
22E0D6    260         LD    (BYTE),HL
CD849D    270         CALL  CHRGRD
CD909D    280         CALL  DISCH
3E7F      290         LD    A,127
210218    300         LD    HL,6146
CD4D00    310         CALL  77
3E31      320         LD    A,49
210C18    330         LD    HL,6156
CD4D00    340         CALL  77
3E30      350         LD    A,48
210B18    360         LD    HL,6155
CD4D00    370         CALL  77
3E30      380         LD    A,48
210A18    390         LD    HL,6154
CD4D00    400         CALL  77
CD779D    410         CALL  RAM2V
3E82      420         LD    A,130
```

```
218B18    430          LD      HL,6283
CD4D00    440          CALL    77
00        450 LOOP:    NOP
3E07      460          LD      A,7
CD4101    470          CALL    321
CB57      480          BIT     2,A
C8        490          RET     Z
3E08      500          LD      A,8
CD4101    510          CALL    321
CB47      520          BIT     0,A
CAEC9D    530          JP      Z,SWITCH
CB7F      540          BIT     7,A
CA3D9D    550          JP      Z,RIGHT
CB77      560          BIT     6,A
CAFB9C    570          JP      Z,CDOWN
CB6F      580          BIT     5,A
CAD89C    590          JP      Z,CUP
CB67      600          BIT     4,A
CA1F9D    610          JP      Z,LEFT
3E04      620          LD      A,4
CD4101    630          CALL    321
CB6F      640          BIT     5,A
CA8B9E    650          JP      Z,PREV
CB5F      660          BIT     3,A
CA329E    670          JP      Z,NEXT
C39F9C    680          JP      LOOP
3AE8D6    690 CUP:     LD      A,(VER)
3D        700          DEC     A
CA9F9C    710          JP      Z,LOOP
32E8D6    720          LD      (VER),A
2AE0D6    730          LD      HL,(BYTE)
2B        740          DEC     HL
22E0D6    750          LD      (BYTE),HL
2AE6D6    760          LD      HL,(CURSOR)
012000    770          LD      BC,32
A7        780          AND     A
ED42      790          SBC     HL, BC
22E6D6    800          LD      (CURSOR),HL
CD909D    810          CALL    DISCH
C39F9C    820          JP      LOOP
3AE8D6    830 CDOWN:   LD      A,(VER)
D608      840          SUB     8
CA9F9C    850          JP      Z,LOOP
C609      860          ADD     A,9
32E8D6    870          LD      (VER),A
2AE0D6    880          LD      HL,(BYTE)
23        890          INC     HL
22E0D6    900          LD      (BYTE),HL
2AE6D6    910          LD      HL,(CURSOR)
```

```
012000    920          LD      BC,32
09        930          ADD     HL,BC
22E6D6    940          LD      (CURSOR),HL
CD909D    950          CALL    DISCH
C39F9C    960          JP      LOOP
3AE9D6    970 LEFT:    LD      A,(HOR)
3D        980          DEC     A
CA9F9C    990          JP      Z,LOOP
32E9D6    1000         LD      (HOR),A
3AE2D6    1010         LD      A,(CHBIT)
87        1020         ADD     A,A
32E2D6    1030         LD      (CHBIT),A
2AE6D6    1040         LD      HL,(CURSOR)
2B        1050         DEC     HL
22E6D6    1060         LD      (CURSOR),HL
CD909D    1070         CALL    DISCH
C39F9C    1080         JP      LOOP
3AE9D6    1090 RIGHT:  LD      A,(HOR)
D608      1100         SUB     8
CA9F9C    1110         JP      Z,LOOP
C609      1120         ADD     A,9
32E9D6    1130         LD      (HOR),A
3AE2D6    1140         LD      A,(CHBIT)
CB0F      1150         RRC     A
32E2D6    1160         LD      (CHBIT),A
2AE6D6    1170         LD      HL,(CURSOR)
23        1180         INC     HL
22E6D6    1190         LD      (CURSOR),HL
CD909D    1200         CALL    DISCH
C39F9C    1210         JP      LOOP
FFC3A599  1220 IXSTR:  DEFB    255,195,165,153
99A5C3FF  1330         DEFB    153,165,195,255
FF81BDBD  1240         DEFB    255,129,189,189
BDBD81FF  1250         DEFB    189,189,129,255
FF818181  1260         DEFB    255,129,129,129
818181FF  1270         DEFB    129,129,129,255
010800    1280 RAM2V:  LD      BC,8
111004    1290         LD      DE,1040
2AE4D6    1300         LD      HL,(CHRADD)
CD5C00    1310         CALL    92
C9        1320         RET
010800    1330 CHRGRD  LD      BC,8
11D8D6    1340         LD      DE,GRID
2AE4D6    1350         LD      HL,(CHRADD)
EDB0      1360         LDIR
C9        1370         RET
210218    1380 DISCH:  LD      HL,6146
11D8D6    1390         LD      DE,GRID
3E08      1400         LD      A,8
```

```
F5          1410 DISCH1  PUSH   AF
1A          1420         LD     A,(DE)
CB7F        1430         BIT    7,A
CDDB9D      1440         CALL   BIT
CB77        1450         BIT    6,A
CDDB9D      1460         CALL   BIT
CB6F        1470         BIT    5,A
CDDB9D      1480         CALL   BIT
CB67        1490         BIT    4,A
CDDB9D      1500         CALL   BIT
CB5F        1510         BIT    3,A
CDDB9D      1520         CALL   BIT
CB57        1530         BIT    2,A
CDDB9D      1540         CALL   BIT
CB4F        1550         BIT    1,A
CDDB9D      1560         CALL   BIT
CB47        1570         BIT    0,A
CDDB9D      1580         CALL   BIT
13          1590         INC    DE
011800      1600         LD     BC,24
09          1610         ADD    HL,BC
F1          1620         POP    AF
3D          1630         DEC    A
C2989D      1640         JP     NZ,DISCH1
CD779D      1650         CALL   RAM2V
3E7F        1660         LD     A,127
2AE6D6      1670         LD     HL,(CURSOR)
CD4D00      1680         CALL   77
CDE39E      1690         CALL   DELAY
C9          1700         RET
F5          1710 BIT:    PUSH   AF
CAE49D      1720         JP     Z,BIT1
3E80        1730         LD     A,128
C3E69D      1740         JP     BIT2
3E81        1750 BIT1:   LD     A,129
CD4D00      1760 BIT2:   CALL   77
F1          1770         POP    AF
23          1780         INC    HL
C9          1790         RET
00          1800 SWITCH  NOP
2AE0D6      1810         LD     HL,(BYTE)
7E          1820         LD     A,(HL)
47          1830         LD     B,A
3AE2D6      1840         LD     A,(CHBIT)
4F          1850         LD     C,A
3E01        1860         LD     A,1
CB79        1870 SWIT1:  BIT    7,C
C2059E      1880         JP     NZ,SWIT2
CB00        1890         RLC    B
```

```
CB01      1900            RLC     C
3C        1910            INC     A
C3F89D    1920            JP      SWIT1
CB78      1930 SWIT2:     BIT     7,B
CA0F9E    1940            JP      Z,SWIT3
CBB8      1950            RES     7,B
C3119E    1960            JP      SWIT4
CBFF      1970 SWIT3:     SET     7,A
3D        1980 SWIT4:     DEC     A
CA1A9E    1990            JP      Z,SWIT5
CB08      2000            RRC     B
C3119E    2010            JP      SWIT4
78        2020 SWIT5:     LD      A,B
77        2030            LD      (HL),A
CDF29E    2040            CALL    GRDCHR
CD909D    2050            CALL    DISCH
CDE39E    2060 WAIT:      CALL    DELAY
3E08      2070            LD      A,8
CD4101    2080            CALL    321
CB47      2090            BIT     0,A
CA229E    2100            JP      Z,WAIT
C39F9C    2110            JP      LOOP
00        2120 NEXT:      NOP
3AE3D6    2130            LD      A,(CURCHR)
D67E      2140            SUB     126
CA9F9C    2150            JP      Z,LOOP
C67F      2160            ADD     A,127
32E3D6    2170            LD      (CURCHR),A
2AE4D6    2180            LD      HL,(CHRADD)
010800    2190            LD      BC,8
09        2200            ADD     HL,BC
22E4D6    2210            LD      (CHRADD),HL
CD849D    2220            CALL    CHRGRD
CD909D    2230            CALL    DISCH
210C18    2240            LD      HL,6156
CD4A00    2250            CALL    74
D639      2260            SUB     57
CA639E    2270            JP      Z,NEXT1
C63A      2280            ADD     A,58
CD4D00    2290            CALL    77
C39F9C    2300            JP      LOOP
3E30      2310 NEXT1:     LD      A,48
CD4D00    2320            CALL    77
210B18    2330            LD      HL,6155
CD4A00    2340            CALL    74
D639      2350            SUB     57
CA7B9E    2360            JP      Z,NEXT2
C63A      2370            ADD     A,58
CD4D00    2380            CALL    77
```

```
C39F9C    2390          JP      LOOP
3E30      2400 NEXT2:   LD      A,48
CD4D00    2410          CALL    77
210A18    2420          LD      HL,6154
3E31      2430          LD      A,49
CD4D00    2440          CALL    77
C39F9C    2450          JP      LOOP
00        2460 PREV:    NOP
3AE3D6    2470          LD      A,(CURCHR)
3D        2480          DEC     A
CA9F9C    2490          JP      Z,LOOP
32E3D6    2500          LD      (CURCHR),A
2AE4D6    2510          LD      HL,(CHRADD)
010800    2520          LD      BC,8
A7        2530          AND     A
ED42      2540          SBC     HL,BC
22E4D6    2550          LD      (CHRADD),HL
CD849D    2560          CALL    CHRGRD
CD909D    2570          CALL    DISCH
210C18    2580          LD      HL,6156
CD4A00    2590          CALL    74
D630      2600          SUB     48
CABB9E    2610          JP      Z,PREV1
C62F      2620          ADD     A,47
CD4D00    2630          CALL    77
C39F9C    2640          JP      LOOP
3E39      2650 PREV1:   LD      A,57
CD4D00    2660          CALL    77
210B18    2670          LD      HL,6155
CD4A00    2680          CALL    74
D630      2690          SUB     48
CAD39E    2700          JP      Z,PREV2
C62F      2710          ADD     A,47
CD4D00    2720          CALL    77
C39F9C    2730          JP      LOOP
3E39      2740 PREV2:   LD      A,57
CD4D00    2750          CALL    77
210A18    2760          LD      HL,6154
3E30      2770          LD      A,48
CD4D00    2780          CALL    77
C39F9C    2790          JP      LOOP
3E32      2800 DELAY:   LD      A,50
F5        2810 DEL:     PUSH    AF
3EFF      2820          LD      A,255
3D        2830 DEL1:    DEC     A
C2E89E    2840          JP      NZ,DEL1
F1        2850          POP     AF
3D        2860          DEC     A
C2E59E    2870          JP      NZ,DEL
```

```
C9          2880          RET
010800      2890 GRDCHR   LD      BC,8
ED5BE4D6    2900          LD      DE,(CHRADD)
21D8D6      2910          LD      HL,GRID
EDB0        2920          LDIR
C9          2930          RET
```

Después de haber introducido y comprobado el programa, puedes guardarlo (SAVE) en cinta usando:

```
BSAVE "CAS:DISEÑE",40000,40800
```

El programa ahora puede ser comprobado introduciendo y ejecutando las siguientes líneas:

```
1000 DEF USR = 40000
1005 A = USR(1)
```

Si has tecleado el programa correctamente verás que aparece en pantalla una gradilla de 8*8. En la esquina superior izquierda de la pantalla verás una X. Esta X puede moverse a lo largo de la gradilla con las cuatro teclas de cursor (flecha). Piensa en los 64 cuadritos como si fueran conmutadores; cada uno de ellos puede encenderse o apagarse. (Lee la sección de Binarios para una mayor explicación de esto). Para iluminar o apagar un cuadrito, primero mueve la X a ese cuadrito. Pulsando la barra espaciadora, alterarás la condición de ese cuadrado. Es decir: si estaba iluminado, pulsando la barra espaciadora se apagará; si estaba apagado se iluminará. Justo a la derecha de la gradilla verás ahora la forma del carácter que estás diseñando.

Moviendo la X por toda la gradilla, puedes diseñar un carácter completo. Este programa puede usarse para diseñar 126 caracteres. En la parte superior de la pantalla verás un número; este es el número asociado -el código- del carácter corrientemente mostrado en la gradilla 8*8. Para pasar a cualquier carácter pulsa N (Next=siguiente). Manteniendo pulsada la tecla N recorrerás todos los caracteres hasta 126. Por supuesto, no verás nada interesante hsta que hayas diseñado el aspecto visivo de esos caracteres. Pulsando la tecla P (Previo), te permitirá regresar hasta el carácter con código 1.

Cuando hayas diseñado algunos caracteres y quieras guardarlos en cinta, pulsa la tecla ESCape. Ahora teclea:

```
BSAVE "CAS:CHARS" 41000,42007
```

Te aconsejo que guardes los caracteres dos veces en dos cintas de cassette separadas. Esto te evitará desperdiciar el duro trabajo efectuado si uno de los cassettes se deteriora.

Ahora haré hincapié en el proceso para diseñar tus propios caracteres:

1) Teclea: `CLEAR 200,35999`
2) Teclea: `BLOAD"CAS:"`

Esto cargará el programa diseñador de caracteres.

3) En este paso quizás desees cargar un grupo de caracteres que ya has empezado a diseñar y los has guardado en cinta. Hazlo tecleando:

```
BLOAD"CAS:"
```

4) Introduce y ejecuta las siguiente líneas:

```
1000 DEF USR = 40000
1005 A = USR(1)
```

5) Diseña tus caracteres.
6) Pulsa ESCape.
7) Guarda tus caracteres en cinta usando:

```
BSAVE"CAS:CHARS",36000,38007
```

Habiendo diseñado tus propios caracteres ahora necesitas saber cómo usarlos en tu propio programa. La secuencia para hacer esto es como se muestra a continuación.

1) Teclea: CLEAR 200,35999
2) Teclea: SCREEN 1
3) Carga tu carácter usando:

BLOAD"CAS:"

Los 126 caracteres ahora están almacenados en la Memoria de Escritura-Lectura; sin embargo, para usarlos necesitas trasladarlos hasta la VRAM. El siguiente programa en BASIC transferirá esos caracteres dentro de VRAM:

```
10 FOR A = 0 TO 2007
20 VPOKE(TIPOCOM + A), PEEK (36000 + A)
30 NEXT A
40 STOP
```

El programa en BASIC anterior pondrá los caracteres a partir de una dirección por la cual el primer carácter que diseñaste tendrá un código de 251. Son estos códigos con los que utilizas VPOKE dentro del mapa de la VRAM para hacer que los caracteres aparezcan en la pantalla.

No cambies el modo de pantalla cuando tus caracteres estén ya en la VRAM; en otro caso se reescribirán con el carácter estipulado como normal en el ordenador.

El siguiente programa en BASIC te mostrará todos los caracteres diseñados que están actualmente en la VRAM:

```
10 FOR A = 126 TO 251
20 VPOKE(PIZACOM + A),A
30 NEXT A
40 STOP
```

Puedes usar los caracteres en tus propios programas en BASIC o en Código Máquina.

Aunque el diseñador de carácter sólo permite que diseñes 126 caracteres, el siguiente método te permitirá rediseñar y usar 252 caracteres:

1) CLEAR 200,35999
2) Carga el diseñador de carácter usando:

BLOAD"CAS:"

3) Teclea y ejecuta las siguientes líneas:

```
1000 DEF USR = 40000
1005 A = USR(1)
```

4) Diseña 126 caracteres.
5) Pulsa ESCape.
6) Guarda los caracteres en cinta usando lo siguiente:

```
BSAVE"CAS:SET1",36000,38007
```

7) Teclea RUN.
8) Diseña otros 126 caracteres.
9) Pulsa ESCape.
10) Guarda los caracteres en cinta usando lo siguiente:

```
BSAVE"CAS:SET2",36000,38007
```

Cuando desees usar los 252 caracteres en tu programa debes hacer lo siguiente:

1) `CLEAR 200,35999`
2) `BLOAD"CAS:SET1"`

3) Introduce y ejecuta el siguiente programa:

```
10 FOR A = 0 TO 2007
20 VPOKE(TIPOCOM + A + 1008),
   PEEK (36000 + A)
30 NEXT A
40 STOP
```

El programa anterior colocará los 126 caracteres del repertorio 1 dentro de las direcciones en la VRAM donde sus códigos sean 126 a 251.

4) `BLOAD"CAS:SET2"`
5) Cambia la línea 20 del programa en BASIC de arriba por:

```
20 VPOKE(TIPOCOM + A), PEEK (A + 36000)
```

6) Ahora teclea RUN. Esto transferirá el repertorio 2 de caracteres a las direcciones de la VRAM donde sus códigos serán 0 a 125. Habiendo hecho esto puede que encuentres lo que hay en la pantalla difícil de leer, a menos que diseñases algunas letras y números en las posiciones correctas en el repertorio de caracteres 2.

Obviamente, si vas a escribir un programa que usa muchos de los caracteres rediseñados, es mejor dejar el actual diseño de los caracteres hasta que hayas completado el programa.

DISEÑADOR DE FIGURAS

El diseñador de figuras puede también usarse para diseñar el tipo o forma visiva de las Figuras. El siguiente procedimiento te mostrará cómo diseñar el tipo de 32 Figuras:

1) CLEAR 200,35999
2) Cargar el programa diseñador de caracteres tecleando:

```
BLOAD"CAS:"
```

3) Introducir y ejecutar las siguientes dos líneas:

```
1000 DEF USR = 40000
1005 A = USR(1)
```

4) Diseña la forma visiva de las 32 Figuras en las posiciones donde normalmente diseñarías los caracteres 1 a 32.
5) Pulsa ESCape.
6) Guarda las 32 Figuras en cinta tecleando lo siguiente:

```
BSAVE"CAS:SP32",36000,38007
```

Cuando desees usar las Figuras diseñadas, sigue este procedimiento:

1) CLEAR 200,35999
2) Carga las formas de las Figuras tecleando lo siguiente:

```
BLOAD"CAS:SP32"
```

3) Las formas de las figuras ahora se transfieren a sus posiciones correctas en la VRAM con el siguiente programa:

```
10 FOR A = 0 TO 255
20 VPOKE(FACHACOM + A),
   PEEK (A + 36000)
30 NEXT A
40 STOP
```

Ahora puedes usar VPOKE con algunos de los atributos o Portores de las Figuras e intentar hacer que las Figuras aparezcan en pantalla.

NOTA DEL AUTOR

Espero que haya conseguido lo que me propuse hacer, que no fue otra cosa que darte una ligera introducción dentro del mundo de la programación en Código Máquina. Deberías haber descubierto si tú y el Código Máquina formáis una buena pareja. Hay muchos códigos de operación del Z80 con gran detalle. Cuando empieces a escribir tus programas en Código Máquina quizás te enfrentes ante algunos problemas que no estés capacitado para resolver. Si me escribes a través de los editores de este libro intentaré ayudarte encantado (¡pero no te contestaré si no me mandas el sobre con el sello para la respuesta!).

Por favor, no me escribas cartas tales como "He encontrado una forma más rápida de crear la explosión en el programa Invasor Espacial". Cartas así se pasarán directamente a la basura.

Steve Webb
61-63 Portobello Road
London W11
January 1985

Respuestas

1) La parte alta de 45621 es 178, la parte baja es 53.
2) Si la parte baja de un número es 31 y la parte alta es 64, el número es 16415.
3) Si la celdilla 40000 contuviese 5 dec y la celdilla 4001 contuviese 15 dec después de la instrucción LD HL, (4000), HL contendrá el valor 3845.
4) Si HL contiene 35621, después de la instrucción LD(40000),HL la celdilla 40000 contendrá 37 y la celdilla 40001 contendrá 139.
5) El equivalente decimal de E3h es 227.
6) Si FBh es la parte alta de un número y CBh es la parte baja, el número es 64459.

MSX

CODIGO MAQUINA
PROGRAMACION PRACTICA

La llegada del standar MSX, marca un paso significativo dentro del mundo de los ordenadores domésticos. Ofrece infinidad de excitantes oportunidades para programadores y usuarios.

Los ordenadores MSX tienen muchas características avanzadas tales como las figuras móviles y definibles por el usuario, y su sonido. Si deseas obtener el máximo beneficio de estas facilidades, necesitarás programar en código máquina.

Este libro, supone que no tienes conocimientos previos sobre código máquina. Empieza explicando las equivalentes en código máquina, de las principales instrucciones en Basic, tales como IF, FOR/NEXT, PRINT, GOTO, GOSUB, etc... Continúa con la descripción muy detallada de las rutinas individuales de un simple juego: "INVASOR DEL ESPACIO", y cómo se enlazan estas rutinas que puedes aprovechar en tus programas en Basic.

También hay, a lo largo del libro, numerosas preguntas que auto-evaluan tus conocimientos sobre Programación en código máquina.

- Aprende lo fácil que es incorporar rutinas en código máquina dentro de tus programas en Basic.
- Usa el código máquina para crear sonido y gráficos
- Diseña tus propios caracteres y "escenarios" con dos programas fáciles de usar.

Tanto si simplemente deseas incluir rutinas para acelerar y mejorar tus programas en Basic, como si quieres un programa completo y potente en código máquina, encontrarás este libro extremadamente útil. En él está suprimida la teoria que dificulta el código máquina, y se presenta de una forma práctica y divertida de aprenderlo.