

CURSO DE DESPROTECCIÓN **DE PROGRAMAS**

CAPITULO 1

1. NOCIONES GENERALES.

Todo lo expuesto en este breve e inexacto texto estará referido al MSX-1 (salvo que se diga lo contrario) y al casete. No obstante, podrá aplicarse al MSX-2 si antes los usuarios de estos ordenadores introducen en ellos los famosos “pokes mágicos” que permiten cargar programas de MSX-1 en MSX-2 sin problemas.

La utilización de la unidad de disco para cargar y grabar programas desprotegidos, en MSX-1, plantea el problema adicional de que, como sabemos, la unidad de discos resta unos 4Kbs, al estar conectada, a la memoria que se utiliza para el Basic (y por tanto a la memoria que utiliza el Basic MSX para cargar programas), por ello al utilizar la unidad de discos deberá procurarse que el programa a cargar no ocupe direcciones de memoria que utiliza dicha unidad para su control y funcionamiento. En los MSX-2, este problema puede ser fácilmente subsanado si se conocen y se saben utilizar los subslots (o slots expandidos) no presentes normalmente en el MSX-1.

Antes de continuar, definiremos unos cuantos **CONCEPTOS BÁSICOS:**

- **ROM:** (Read Only Memory). Memoria de solo lectura. La ROM puede ser leída pero no podrás escribir nada en ella.
- **RAM:** (Random Acces Memory). Memoria de acceso aleatorio o memoria de lectura-escritura.
- **SLOT:** Banco de memoria de 64 kb (numerados de 0 a 3).
- **PAGINA:** Cada uno de los bloques de memoria de 16 kb en que se divide un slot (numerados de 0 a 3).
- **S.O.:** (Sistema Operativo MSX). Situado en la página 0 del slot 0.

Además de estos conceptos básicos, sería útil, que no imprescindible, que conocieseis algo de CM o ensamblador; además también sería útil que conocieseis el sistema numérico binario y el hexadecimal... Conforme se vaya desarrollando este texto iré explicando otros conceptos que son necesarios entender.

2. LA MEMORIA MSX

Antes de comenzar con la desprotección en sí, nos detendremos un poco en otro tema capital que es necesario estudiar; la configuración de la memoria de nuestro MSX. Conocer como se dispone y se maneja la memoria es IMPRESCINDIBLE para poder desproteger programas.

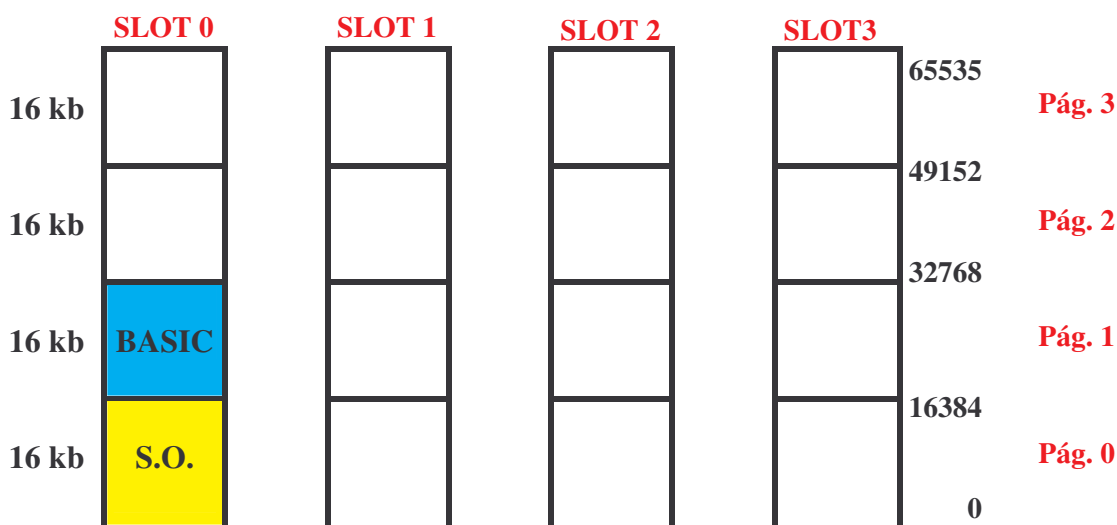
Lo primero que deberemos hacer es lo siguiente:

PRINT INP(&HA8) (Pulsar ENTER).

A lo que nuestro MSX responderá con un número (de 0 a 255, que es lo que se llama un BYTE), como por ejemplo 80 (Panasonic CF2700). Este número deberemos anotarlo puesto que es de gran importancia....!!!!

La memoria de nuestro MSX se configura en BANCOS, slots de memoria de 64 kb cada uno; estos bancos a su vez se dividen en 4 páginas de memoria de 16 kb cada una. Por supuesto tanto los slots como las páginas están numeradas. Todo esto lo entenderás mejor si te fijas en el siguiente esquema:

Esquema 1:



Como observaréis por el gráfico, cada página de memoria tiene 16 kb numerados de la siguiente forma:

- **Página 0:** Desde **0** a **16383** (**&H3FFF**).
- **Página 1:** Desde **16384** a **32767** (**&H4000** a **&H7FFF**).
- **Página 2:** Desde **32768** a **49151** (**&H8000** a **&HBFFF**).
- **Página 3:** Desde **49152** a **65535** (**&HC000** a **&HFFF**).

Para los más despistadillos, os recuerdo que estos números son direcciones de memoria, como es lógico. ☺ Alguno se preguntará por qué he colocado el **S.O.** y el **BASIC** en las **páginas 0 y 1 del slot 0** en vez de ponerlos en cualquier otro lugar del esquema: pues bien, este, y otros hechos, son los que garantizan la tan traída y llevada (y puesta en duda) **compatibilidad del sistema MSX**. O sea, los fabricantes de ordenadores de la norma MSX pueden incorporar en sus aparatos uno, dos, tres o cuatro slots (no hay un número obligatorio ni fijo, aunque siempre como es lógico habrá al menos un slot), pero lo que todos hacen es colocar el **S.O.** y el **BASIC** en las páginas antes citadas del slot 0, que dicho sea de paso son la ROM del MSX. Esto es así, entre otras cosas, porque siempre tiene que existir un slot como mínimo y este debe ser el slot 0. ☺

Tal vez os preguntareis que es lo que hay en los espacios en blanco de los slots del esquema 1; lógicamente si estamos hablando de bancos de memoria, en estos lugares no podrá haber otra cosa que no sea memoria. Podrá haber memoria ROM (programas Agenda de algunos SONY por ejemplo), pero lo más normal es que haya memoria RAM, sobre todo si nuestro MSX es de 64 kb o más.

Surge entonces la inevitable pregunta: ¿todos los espacios en blanco están ocupados por RAM?. , a la que sigue la inevitable y ambigua respuesta: NO NECESARIAMENTE, pero puede ocurrir, sobre todo en los MSX-2 (en el MSX-1 no suele ocurrir). Partimos de la hipótesis , no lo olvidéis, de que nuestro MSX tiene 4 slots, cosa que no tiene porque ser cierta, si los slots 2 y 3 no existen por ejemplo no cabe entonces preguntarse si hay o no RAM en estos slots.

Se plantea entonces otra inevitable pregunta (y esto ya parece el Tiempo es Oro....): ¿cómo podemos saber que páginas y que slots contienen memoria RAM? (que es la memoria de almacenamiento de datos por si alguno no había caído en ello....). En definitiva, queremos saber que slots ha incorporado el fabricante de nuestro ordenador, porque si una página de un slot contiene RAM por economía las demás páginas de ese slot también van a contener memoria RAM (excepto en el slot 0 como ya sabemos). ☺ No obstante, no hagáis mucho caso de esta afirmación porque no es del todo cierta, sobre todo en lo referido a los MSX-2, a los cartuchos de ampliación de memoria, etc, etc... No es lógico, ni barato, incorporar un slot que contenga sólo una página útil (que tenga RAM), lo normal en el caso de los MSX-1 es incorporar slots completos (con 64 kb de RAM).

Si alguien se ha perdido - no creo-, que vaya buscándose porque a partir de ahora comienza lo realmente importante: LA PRÁCTICA.

Como todos sabemos, nuestro MSX sólo puede gestionar 64 kb de memoria a la vez, ya sea un MSX-1 o MSX-2, y esto es así porque el “cerebro” de los MSX’s es el conocido microprocesador **Z80A**.

Estos 64 kb pueden estar repartidos entre las distintas páginas de los slots. Para que los programadores y el propio S.O. distribuyan a conveniencia las páginas de memoria, los creadores del MSX incluyeron en el un **PUERTO** (o CANAL) de **entrada/salida: &HA8**, que al leerlo nos informa sobre las páginas de memoria que están activadas y al escribir un dato en él (mandar un dato) se modifica la configuración de las mismas.

¿Cómo sabremos qué dato debemos enviar a ese puerto para modificar la paginación de la memoria?, o lo que es lo mismo, ¿cómo interpretaremos el dato leído desde este puerto para saber cual es la actual paginación de memoria?. Para tales motivos se creó la siguiente convención:

00→ Indica **SLOT 0**.
01→ Indica **SLOT 1**.
10→ Indica **SLOT 2**.
11→ Indica **SLOT 3**.

Dato a Leer/Escribir:

	Bits 6 y 7	Bits 4 y 5	Bits 2 y 3	Bits 0 y 1	
&B	0 0	0 0	0 0	0 0	Byte 0..255
	P á g. 3	P á g. 2	P á g. 1	P á g. 0	

Bits **0 y 1**→ **Página 0**.
Bits **2 y 3**→ **Página 1**.
Bits **4 y 5**→ **Página 2**.
Bits **6 y 7**→ **Página 3**.

Puede parecer muy complicado, pero en realidad no lo es tanto, veámoslo con un **ejemplo**:

Supongamos que queremos activar las **páginas 0 y 1** en el **slot 0** y las **páginas 2 y 3** en el **slot 1**. El dato a enviar por el **puerto A8** sería el siguiente:

&B	0 1	0 1	0 0	0 0	80 decimal
	Pág. 3	Pág. 2	Pág. 1	Pág. 0	
	S L O T 1		S L O T 0		

Esta configuración de memoria se activaría con **OUT &HA8, 80**, pero no os apresuréis a activarla porque lo más seguro es que se os “cuelgue” el ordenador (cosa que suele ocurrir mucho las primeras veces que se trastea con la memoria), a no ser que se trate de un PANASONIC CF2700, como es mi caso.

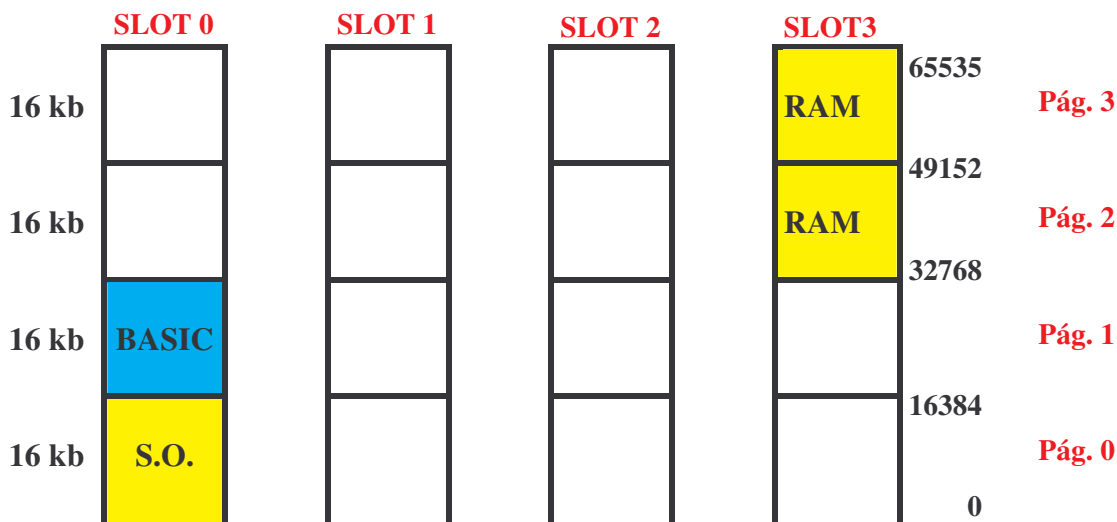
¿Recordáis el número del principio?. ¿Si?. Me cuesta creerlo...Coged ahora ese dato y convertirlo a binario con la instrucción: **PRINT BIN\$(X)**. Supongamos que al hacerlo nos da esto:

```

&B      1 1      1 1      0 0      0 0
        Pág. 3    Pág. 2    Pág. 1    Pág. 0
        S L O T 3      S L O T 0

```

Gráficamente:



Con este dato sabemos, en este ejemplo, que las **páginas 0 y 1** están activadas en el **slot 0**, y que las **páginas 2 y 3** contienen **RAM** y están en el **slot 3**. Con esta configuración seguro que estamos trabajando en BASIC MSX (a la fuerza, porque si no, ¿cómo íbamos a poder hacer PRINT BIN\$(X)...? ☺ Además, con este dato podemos suponer, sin temor a equivocarnos mucho que el **slot 3** contendrá **64 kb** de **RAM**.

Pero, ¿qué sucederá si activásemos las **páginas 0 y 1** en otro **slot distinto del 0?**, o lo que es lo mismo, ¿qué sucede si desactivamos el **BASIC** y el **S.O.**? Desde luego si desactivamos el BASIC desde el BASIC o el S.O. desde el BASIC, va a suceder lo más obvio: nos quedaremos colgados... Por ello, deberemos hacerlo con el tan temido CM y con sumo cuidado, dicho sea de paso. Yo personalmente prefiero desactivar el S.O., página 0 del slot 0, lo menos posible pues en él se encuentran rutinas CM que son importantes y muy útiles para el programador (sobre todo si eres un aficionado como lo soy yo).

A la **memoria RAM** que se encuentra situada en las famosas **páginas 0 y 1**, se la denomina **RAM OCULTA**, porque normalmente, con el BASIC activo, permanece dicha memoria oculta al usuario.

Ahora viene la primera superpregunta: Hasta el momento hemos supuesto que si un slot tenía una o más páginas de RAM ese slot contenía 64 kb de RAM, pero como ya apunté antes esto NO tiene porque ser así... Se plantea entonces la superpregunta, de difícil respuesta: ¿cómo podemos saber entonces que páginas de los distintos slots contienen memoria ROM o RAM? (nos interesa sólo la RAM).

Para suministrarnos esa información, usaremos los conocidos **ANALIZADORES DE SLOTS**. Un programa de este tipo no es más que un conjunto de rutinas CM que analizan la configuración de memoria de nuestro MSX hasta dar con un slot lleno de RAM (si lo hay), y entonces escribe ese dato (el de la configuración de memoria TODO RAM) en algún lugar de la memoria para que pueda ser leído después. Ésta última parte, la de escribir el dato, no está presente en todos los analizadores, pero sí en los que vamos a usar: los de **TOPO SOFT, DINAMIC y GREMLIM**.

Crear un ANALIZADOR DE SLOTS, es una tarea que aunque sencilla requiere de ciertos conocimientos de CM y del MSX que puede que algunos no tengáis, por eso vamos a usar los programas comerciales para extraer de ellos el analizador de slots que usaremos.

Cojamos por ejemplo, el programa original **BLACK BEARD** de TOPO. Tras cargar el logotipo de la empresa, hacemos *CTRL.+BREAK*, y tecleamos **LOAD"CAS:"**, con lo cual aparecerá ante nosotros esto:

```
10 COLOR 1, 1, 1: SCREEN 2
20 BLOAD"CAS:LOAD.BIN"
30 BLOAD"CAS:TEST.BIN"
40 DEFURS=55000!
50 A=USR(0)
```

El bloque de datos llamado **TEST.BIN** es nuestro objetivo. Procederemos a copiarlo con un "copión" o lector de cabeceras, en una cinta aparte. Sus direcciones para copiarlo son:

- Dirección de **INICIO** en memoria: **&HC350**
- Dirección **FINAL** del programa...: **&HC44C**
- Dirección de **EJECUCIÓN**.....: **&HC350**

Una vez copiado el analizador de slots de TOPO, lo cargaremos con **BLOAD"CAS:", R**, y veremos que al parecer "no ha sucedido absolutamente nada". En realidad el programa ya se ha ejecutado y nos ofrece su valiosa información en memoria:

- En la dirección **&HE290**, tenemos el dato (byte) que contiene la configuración normal de memoria, o sea, la que tiene el ordenador nada más encenderlo (el dato que leísteis al principio), a la que llamaremos **CONFIGURACIÓN ROM/RAM**.
- En la dirección **&HE292**, tenemos el dato (byte) que nos informa sobre que slot contiene los 64 kb de RAM. A esta configuración la llamaremos **TODO RAM**.

Para interpretar correctamente esta información debéis leer los datos con **PEEK(&HE290)** y **PEEK(&HE292)** y pasar el dato a binario para interpretarlo como ya sabemos.

Ejemplo: Imaginad que tras ejecutar el **TEST** leemos el dato contenido en **&HE292** y obtenemos **85** (en decimal), la interpretación de este dato se haría así:

$$\begin{array}{ccccccc} \text{\textcolor{blue}{&B}} & & \text{\textcolor{blue}{0}} & \text{\textcolor{blue}{1}} & & \text{\textcolor{blue}{0}} & \text{\textcolor{blue}{1}} & & \text{\textcolor{blue}{0}} & \text{\textcolor{blue}{1}} & & \text{\textcolor{blue}{0}} & \text{\textcolor{blue}{1}} & & = 85 \\ & & \text{P á g. 3} & & \text{P á g. 2} & & \text{P á g. 1} & & \text{P á g. 0} & & & & & & & \\ & & \text{S L O T} & & \text{1} & & \text{S L O T} & & \text{1} & & & & & & & \end{array}$$

Gráficamente tenemos:

SLOT 1	
RAM	Pág. 3
RAM	Pág. 2
RAM	Pág. 1
RAM	Pág. 0

Sabremos entonces que el **slot 1** con tiene **64 kb de RAM**, memoria más que suficiente para cargar los programas “turbo” (“rayitas”) que al ser conversiones del SPECTRUM ☹ suelen tener de 40 a 44 kbs.

Vamos ahora a extraer el Analizador de Slots de otro programa original, pero ahora de **DINAMIC**. El elegido ahora es **NAVY MOVES 1**. Cargamos el programa normalmente, pero estamos muy atentos para que al salir la portada hagamos **CTRL.+STOP**. Cargamos el siguiente bloque de la cinta, el llamado **RELOC**, y a continuación cargamos el bloque llamado **SLOTS** (es muy amable por su parte el que les pongan nombres tan significativos a los bloques). ☺

Este **SLOTS**, lo grabamos junto con el **TEST.BIN** de TOPO SOFT, y después lo cargamos con **BLOAD”CAS:”,R**. En apariencia “nada ocurre”, pero en realidad sucede que:

- En **&HDDD5** tenemos el dato llamado **ROM/RAM**.
- En **&HDDD7** tenemos el dato llamado **TODO RAM**.

Las direcciones de este programa CM, para quien no tenga copión, son:

- Dirección de **INICIO** → **&H9000**
- Dirección de **FIN** → **&H9092**
- Dirección de **EJECUCIÓN** → **&H9000**

Este analizador de slots es más corto que el de TOPO, lo cual no es muy importante, y tiene exactamente la misma longitud y direcciones que el analizador de GREMLIM

Graphics (que es el que yo suelo usar). Sucede con el analizador de slots de GREMLIM que todavía no podéis copiarlo puesto que, aunque muchos de sus programas lo incluyen (Future Knight, TrailBlazer, Avenger, Krakout, Auf Monty,), está colocado justamente al principio de los bloques protegidos. Y ¡todavía no sabemos como desproteger programas...! ☹ ☹

Antes de acabar con este primer capítulo, y con el tema de la memoria, voy a daros algunos “**consejillos generales**”, extraídos de mi propia experiencia personal, en cuanto al **uso de los slots y de las páginas de memoria:**

- a) No desactivar-activar NUNCA paginas de memoria desde el BASIC (el cuelgue es casi seguro). ☹
- b) Al desactivar-activar páginas de memoria no puede nunca desactivarse la página de memoria en la que se encuentra el programa CM que está trabajando. Así, por ejemplo, si nuestro programa está en la página 3 del slot 0 no podemos desactivar esta página del slot 0 y activarla en otro slot mientras el programa se esté ejecutando porque el cuelgue será total. ☹ Por lo tanto, la única manera de desactivar esta página del slot 0 es hacerlo con un programa situado en las páginas 0, 1 ó 2 activas (de cualquier slot).
- c) Cuidado con cambiar la página 3 de ubicación sin desactivar previamente el S.O. Recuerda que esta página contiene la RAM del S.O. y la pila, que son necesarios cuando éste está conectado (¡¡¡mucho ojo con la pila!!!).
- d) Procurad al trabajar en CM no desactivar el S.O., salvo que sea imprescindible, porque al hacerlo os quedaréis sin las utilísimas rutinas CM que éste contiene.
- e) Si es posible, ampliad vuestros conocimientos sobre este tema con cualquier libro que lo trate, o leyendo un artículo aparecido en la MSX-CLUB titulado EL ORDENADOR POR DENTRO, de Ramón Salas, o bien si podéis conseguid la INPUT-MSX nº 1, algo difícil, y leed un artículo sobre el tema que incluye.

Antes de pasar a la desprotección en sí de programas, conviene que hayáis entendido perfectamente todo lo anteriormente expuesto. Si es necesario que lo leáis 2 ó 3 veces (no lo creo), hacedlo, hasta que creáis que lo hayáis comprendido bien.

Para ver si lo has entendido todo, todo, TODO, te propongo unos sencillos ejercicios (que convendría que hicieses pues con la prácticas se aprende todo mucho mejor).

EJERCICIOS PROPUESTOS:

1. Calcula el dato que debemos enviar al **puerto A8** para activar las 4 páginas de memoria en los **slots 0, 1, 2 y 3**.
2. Activa a continuación las siguientes configuraciones :
 - a) Páginas **0 y 3**, en slot **2**.
Páginas **1 y 2**, en slot **3**.
 - b) Páginas **0 y 2**, en slot **1**.
Páginas **1 y 3**, en slot **3**.
(la página 3 no se activará, su valor nos será desconocido: XX).
3. ¿Qué significado puede tener el que el slot **0** de su ordenador contenga lo siguiente?. (Una pista: HIT-BIT de SONY).
 - Páginas **0 a 2**: ROM.
 - Página **3**: RAM.
4. Con la configuración de memoria del ejercicio anterior, y suponiendo que estamos en el BASIC, ¿cuánta memoria aproximadamente le quedará libre para sus programas en BASIC?. (Tenga en cuenta que la **RAM del Sistema** ocupa unos **4 kb** de memoria, y va desde **&HF380** a **&HFFFF**).

¿Podrá almacenar sus programas CM en la página **1**? ¿Y en la página **3**?
¿Por qué?
5. Si queremos hacer un programa CM que desactive las páginas **0 y 1** del slot **3** y las active en otro slot, por ejemplo el **1**, ¿Dónde **NO** deberemos colocar nunca este programa?. ¿Por qué?
6. ¿Cuál sería la cantidad máxima de memoria que podrá tener un MSX-1 si suponemos que los 4 slots primarios contienen RAM en su totalidad?

¿Sería posible tener, según lo explicado, slots de más de 64 kb?. ¿Por qué?
7. Si al cargar y ejecutar el **Analizador de Slots de DINAMIC**, por ejemplo, resulta que el dato que hay en **&HDDD5** es igual al contenido en **&HDDD7**, y lo hemos cargado desde el BASIC, ¿qué podrás deducir de este hecho?
8. ¿Por qué crees que los programadores se molestan en hacer unos programas que simplemente anotan un par de datos en la memoria?. ¿Qué utilidad tendrán (piénsalo bien) estos 2 datos para el programador?
9. ¿Es PROBABLE, que tras ejecutar el **Analizador de Slots de DINAMIC**, la posición **&HDDD5** contenga el valor **80**, y la posición **&HDDD7** contenga el valor **95**?. ¿Por qué?

- 10.** Si a un MSX-1 de **64 kb**, le conectamos una unidad de disco: ¿cuánta memoria aproximada nos quedará para cargar programas desde el BASIC?. (La memoria para cargar programas desde el BASIC comienza lógicamente en **&H8000** = 32768).

P.D. En primer lugar, quiero realizar una aclaración de suma importancia... Este curso fue creado originalmente por FRANCISCO FUERTES LORENZO. El formato original del mismo son fotocopias escritas a máquina, y que yo JOSÉ MANUEL SOTO, e intentado pasar a pdf para luego aplicarle soft de OCR, pero viendo los desastrosos resultados he optado por teclearlo todo letra a letra en formato WORD, respetando fielmente el original...

Para cualquier aclaración, comentario, sugerencia, etc, etc... no dudéis en escribirme. Me gustaría que todo aquel que tenga conocimientos sobre el tema, y pueda contribuir a mejorar este curso, ampliarlo, corregirlo, etc, que se anime y me escriba. También me gustaría me escribierais para darme vuestra opinión sobre el mismo, si lo habéis leído, y tenéis intención de seguirlo.

jose_manuel@mixmail.com

CURSO DE DESPROTECCIÓN DE PROGRAMAS

CAPITULO 2

0. SOLUCIONES EJERCICIOS TEMA 1.

1. Slot 0 – Dato &B 00 00 00 00 = 0
Slot 1 – Dato &B 01 01 01 01 = 85
Slot 2 – Dato &B 10 10 10 10 = 170
Slot 3 – Dato &B 11 11 11 11 = 255
2. a) &B 10 11 11 10 = 190
b) &B XX 01 11 01 = ¿?
3. Puede significar que nuestro ordenador es un SONY por lo que lleva un programa en la ROM (una Agenda, programa de gestión, etc.). Lo que sin duda significa esta configuración de memoria es que el Slot 0 tan sólo contiene 16 kbs de RAM.
4. Quedarán libres para el BASIC, para almacenar programas, poco más de 12 kbs de memoria (16 kbs de RAM menos los 4 kbs que ocupan la RAM del Sistema).
5. Obviamente no lo colocaremos nunca en esas páginas del Slot 3, páginas 0 y 1; la explicación de este hecho es compleja (es necesario saber código máquina o al menos tener algunos conceptos adquiridos como **PC** = Contador de Programa, microprocesador, y otros.). Una explicación simplista podría ser la de decir que el Z80 no puede desactivar la página donde se encuentra el programa que se está ejecutando porque si lo hiciésemos sucedería que durante unos microsegundos, como mínimo, el Z80 no sabría que hacer a continuación (pues virtualmente no existe programa para él durante ese tiempo), lo que haría que el ordenador se bloqueara (ya que el programa que se estaba ejecutando está ahora en una zona de memoria desactivada). NO hagáis mucho caso de esta explicación pues como digo, la realidad dista bastante de ser tan simple....
6. Si los 4 Slots primarios están llenos de RAM, nuestro MSX tendrá 256 kbs de memoria (esto es sólo una suposición, no os olvidéis nunca del BASIC y del S.O. que son imprescindibles).

Según lo aquí explicado, lo más simple, no sería posible tener Slots de más de 64 kbs porque hemos dicho que un Slot contiene “SIEMPRE” cuatro páginas de memoria de 16 kbs. Esta afirmación no es cierta puesto que un Slot puede tener más de 4 páginas de memoria de 16 kbs, aunque sólo pueda tener cuatro páginas activadas a la vez; esto

ocurre en los MSX2 en los que las páginas de memoria de algunos Slots están ampliadas. Esto significa que dichas páginas se configuran a si mismas como si fuesen auténticos Slots (se les denominan SUBSLOTS), y se subdividen a su vez en otras 4 sub-páginas. Así por ejemplo, la página 0 del Slot 0 puede ser un subslot y contener las subpáginas 0-0, 0-1, 0-2 y 0-3.

El mecanismo de funcionamiento de estos subslots me es desconocido por carecer del equipo necesario para poder estudiarlo, es decir, que como no tengo un MSX-2 no sé como funcionan. ☹ De cualquier forma, debe operar de un modo similar a lo aquí explicado....

7. Deduciré algo tan simple, como el hecho de que mi ordenador no dispone de más cantidad de memoria RAM que la que se encuentra activada con el BASIC (no puede por tanto tener 64 kbs), lo que significa que dispongo de un ordenador de 16 o 32 kbs.
8. Estos programas son necesarios porque el programa a cargar en la memoria deberá desactivar y activar páginas de memoria según lo necesite para su almacenamiento y posterior ejecución, por lo tanto necesitará saber en todo momento dónde hay la suficiente RAM para su almacenamiento.

Los datos ROM/RAM y TODO RAM se almacenan en posiciones fijas de la memoria porque tras cargar cada bloque de datos va a ser necesario “echar mano” de ellos para activar-desactivar las páginas de memoria adecuadas. El programador sabe que es preferible analizar una sólo vez la memoria, guardando los resultados obtenidos de ese análisis, que tener que analizarla cada vez que un bloque de datos es cargado, para así almacenarlo.

9. Es poco probable que ésto ocurra, a no ser que tengamos un MSX-2, puesto que supone un despilfarro hacer un ordenador con tres Slots a “medias”, es decir, es más rentable incorporar Slots completos de 64 kbs de RAM que colocar en nuestro MSX dos Slots (el 1 y el 3 en nuestro ejemplo), que sólo dispongan de 32 kbs de RAM, que estén “a medias”.

Para comprender perfectamente lo expuesto deberíais hacer el gráfico de la memoria de vuestro MSX (¡Si!, ese gráfico de bloques del que tanto hemos hablado), y colocar en su lugar correspondiente tanto la RAM como la ROM de este ejemplo. Cuando lo veas dibujado, seguro que comprendes inmediatamente a que me refiero cuando hablo de Slots “a medias” y de lo “extraño” que parece... No obstante, que no sea probable no significa que no sea posible, es más, en muchos ordenadores MSX-1 el Slot 0 es un Slot “a medias”, (cosa que resulta obvia si has comprendido todo lo dicho anteriormente).
☺

10. La memoria para el BASIC consta de 32 kbs de RAM, en teoría, pero como sabemos unos 4 kbs son necesarios para el S.O. (RAM del Sistema), con lo cual nos quedamos con los 28 kbs típicos del BASIC. A esos 28815 bytes free deberás restarle otros 4 kbs

aproximadamente, que son los que se necesitan para manejar la unidad de discos, con lo que nos quedaremos con unos penosos 24 kbs para el BASIC. Esta, y no otra, es la razón por la que ciertos programas BASIC e incluso programas CM no cargan o no lo hacen correctamente cuando tenemos la unidad de discos conectada!!!! (se sobreentiende que los cargamos de cinta, of course). ☺

NOTA: Estos ejercicios son meramente orientativos, y sus soluciones no tienen porque ser únicas ni exactas. Es sabido que en Informática existen casi infinitas formas de resolver un mismo problema (sobre todo en Programación). No os extrañe que vuestras soluciones no sean iguales a las que yo os propongo... conformaos con que se parezcan un poquito, tan sólo un poquito. ¡Y no olvidéis LA PRÁCTICA!, que es como verdaderamente se aprende.

1. LAS HERRAMIENTAS BÁSICAS.

Teniendo presente que todo lo que voy a explicar a continuación estará referido a la desprotección de programas en cinta, podemos afirmar que el equipo mínimo necesario para realizar tal labor consta de:

- Programa Ensamblador/Desensamblador. Puede utilizarse cualquiera con un mínimo de calidad, aunque yo utilizo y recomiendo el GEN/MON de HISOFT.
- Uno o varios Analizadores de Slots de los que se conozca mínimamente su funcionamiento. Aunque utilizo y recomiendo el de GRENLIM GRAPHICS, por su compatibilidad casi total (podéis utilizar cualquiera).
- Un ordenador MSX con al menos 64 kbs de memoria RAM. En realidad podemos desproteger programas con menos memoria, pero sucede que después no podríamos probarlos... (captáis la lógica, ¿no?). ☺
- Por último, necesitaréis un copión de cinta o un lector de cabeceras para leer las direcciones de Inicio, Final y Ejecución del programa cargador. Obviamente, también necesitaremos un casete (grabadora).

2. BLOAD"CAS:", R

Todos hemos utilizado miles de veces esta instrucción del BASIC MSX. Si nos remitimos a nuestro manual del BASIC podremos leer lo siguiente: "**BLOAD (Binary Load)**, carga programas de Lenguaje Máquina, o los carga y los ejecuta....".

Todos sabemos lo que hace, pero nos hemos preguntado alguna vez ¿cómo lo hace...?. Es decir, como sabemos el BASIC no es más que un Interpretador entre el humano y la máquina (lo único que es capaz de entender nuestro MSX en última instancia es CM). Así pues las instrucciones BASIC son codificadas por el Interpretador BASIC en grupos de instrucciones CM mucho más sencillas, que son las que "entiende" y ejecuta el Z80A.

Alguno estará pensando que vamos a adentrarnos en complicadísimas explicaciones sobre el CM... Despejad esos pensamientos de vuestras calenturientas mentes porque de momento, no nos interesa conocer como funciona a fondo nuestro MSX. Nos conformaremos con tener una levísima idea de cómo almacena habitualmente nuestro MSX la información en casete....

3. CÓMO SE ALMACENAN LOS DATOS EN EL CASETE.

- a) El primer pitido que se escucha es el denominado **HEADER**. Su función es la de advertir al ordenador acerca de la velocidad de grabación del bloque de datos que se va a cargar (1.200 ó 2.400 baudios).
- b) A continuación viene un grupo de 16 bytes llamado **LEADER**, que indica al ordenador el tipo de fichero que se va a cargar y su nombre.

Este Leader puede tener más de 16 bytes, aunque en los bloques grabados con BSAVE, CSAVE, y SAVE nunca va a superar los 16 bytes. Supongo que los creadores del BASIC MSX decidirían, por conveniencia, que el Leader comprensible para el BASIC tuviera siempre 16 bytes. Así pues, tendremos los siguientes **TIPOS DE LEADER:**

1. El **Leader Normal** de un programa grabado con **BSAVE** sería algo así:

DO DO DO DO DO DO DO DO DO DO DO 41 42 43 44 45 46

- Los 10 primeros bytes, cuyo contenido HA DE SER EL MISMO, sirven para informar al ordenador (vía Interprete BASIC, de que se trata de un bloque de datos BINARIOS (**DO**), y en definitiva que deber cargarse con la instrucción Basic **BLOAD**.
- Los 6 últimos bytes son el nombre del programa (ABCDEF en este ejemplo).

2. El **Leader Normal** de un programa grabado con **SAVE** sería algo así:

EA EA EA EA EA EA EA EA EA EA EA 41 42 43 44 45 46

En los ficheros de datos grabados con SAVE, los 10 primeros bytes contienen el dato “EA”, y los 6 últimos bytes como en el primer caso, son el nombre del programa.

3. El **Leader Normal** de un programa grabado con **CSAVE** sería algo así:

D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 41 42 43 44 45 46

En los ficheros de datos grabados con CSAVE, los 10 primeros bytes contienen el dato “D3”, y los 6 últimos bytes como en el primer caso, son el nombre del programa.

Nuestro MSX BASIC, como sabemos, sólo posee 3 instrucciones distintas de carga de bloques de datos desde el casete (MERGE, no puede considerarse una instrucción autónoma puesto que no es más que un LOAD con verificación posterior). Aparte quedan los archivos de datos que podamos crear con OPEN, que por el momento no nos interesan.

De la configuración del LEADER, y de su utilidad y funcionamiento surgen multitud de dudas y cuestiones confusas... De todas ellas, las que más atraerá nuestra atención es la siguiente: ¿Qué sucederá si cambiamos alguno de los 10 primeros bytes del LEADER....?. Imaginemos por ejemplo un Leader como éste:

DO DO FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00

Ante semejante “cacaomaravillao” de bytes sin sentido, nuestro estricto MSX-BASIC no puede más que pedir auxilio desesperadamente, es decir, que el BASIC, NO SERÁ CAPAZ DE RECONOCER SEMEJANTE LEADER, y por tanto, al no reconocerlo como uno de los posibles casos típicos (DO, D3, y EA), para los que está programado a responder, simplemente ignorará este programa. Tanto rollo para decir que si la configuración del Leader varía lo más mínimo, nuestro BASIC NO será capaz de cargar ese programa, **DESDE EL BASIC** naturalmente!!!. Por cierto, para los despistadillos recuerdo que los últimos 6 bytes del Leader SI podrán tomar distintos valores (ya que son el Nombre del Programa!!!!). Los que no pueden alterarse, son los 10 primeros, ¿ok?.

Lógicamente, más de uno se habrá dado cuenta de que si al cambiar el LEADER lo que hacemos es que el programa no pueda ser cargado desde el BASIC, entonces no hemos hecho otra cosa que no sea **proteger dicho programa**. Un **PROGRAMA PROTEGIDO** es por tanto, todo aquel que **no puede ser cargado directamente desde y en BASIC-MSX**. De esta definición simplista y de “mi cosecha”, se sigue que el **DESPROTEGER UN PROGRAMA**, para nosotros, no es mas que **hacer que un programa protegido pueda ser cargado desde y en BASIC-MSX!!!**. ☺

Esta técnica de proteger los programas alterando su Leader no es muy utilizada, tal vez por lo sencillo que resulta el desproteger estos programas. Mucho más habitual es **prescindir totalmente del Leader**, es decir, ¡quitarlo de en medio!, ¡darle el pasaporte!, ¡tirarlo a los tiburones del Hudson!....

- c) Por último, tras el Leader, hay un segundo grupo de datos precedidos por una especie de HEADER (¿Se ha perdido alguien?). Este segundo grupo de datos (Segundo Header) varía dependiendo del tipo de fichero que es está cargando, es decir, si es del tipo **DO, D3 ó EA**.

Para los **Ficheros Binarios**, que son los que realmente nos interesan, este “Segundo Leader”, por llamarlo de algún modo, está formado por 6 bytes que corresponden a:

XX-YY	II-JJ	EE-FF
Dir. Inicio en la RAM del programa	Dir. Final en la RAM del programa	Dir. Ejecución en la RAM del programa.

Es evidente, que este segundo Leader es distinto para cada programa, puesto que cada programa puede tener una dirección de Inicio, Fin o Ejecución distinta. Lo importante no es el dato en si, si no, la forma en como se organizan esos datos. El siguiente esquema, muestra la **Organización Secuencial de los Datos en el Casete:**

Nombre	Longitud Bytes
1. HEADER.	¿ ?
2. LEADER.	16 (diez bytes constantes)
3. Segundo HEADER.	¿ ?
4. Segundo HEADER.	6 (Ficheros Binarios)
5. ↓ Programa.... ↓	↓
6. ↓ Programa... ↓	
.....	

Cualquier alteración de los LEADER supone en último término, **PROTEGER EL PROGRAMA Y QUE EL BASIC NO SEA CAPAZ DE CARGARLO!!! ☺**

4. UN POQUITO DE CÓDIGO MÁQUINA.

Lamentándolo mucho, creo que ha llegado el momento de que aprehendais algo de CM. Es inevitable..., ¡debe ser así...!.

A) LOS REGISTROS DEL Z80.

Los registros son unos dispositivos de metal-óxido-semiconductor que constituyen la forma de memoria más rápida existente (se encuentran en el interior del microprocesador Z80 lógicamente). Los registros internos están habitualmente etiquetados de **0** a **n**, y su función no está definida de antemano (por eso se dice que son **Registros de Propósito General**). Pueden contener cualquier dato utilizado por el programa.

Estos Registros de Propósito General, suelen utilizarse para almacenar datos de 8 bits (un byte, o sea, cantidades numéricas comprendidas entre 0 y 255). En algunos microprocesadores, como el Z80, pueden manipularse dos de estos registros simultáneamente, que llamaremos “**Registros Apareados**”, y que permiten el almacenamiento de magnitudes, datos o direcciones, de **16 bits** (desde **0** a **65.535**).

- Los **Registros de Propósito General** de 8 bits del Z80 se denominan:

B, C, D, E, H y L.

El Z80 dispone de dos bancos de datos, ver esquema de registros, es decir, de dos grupos idénticos de seis registros cada uno. En un momento determinado, sólo pueden usarse seis registros, pero hay instrucciones especiales CM para intercambiar sus contenidos entre los dos grupos. Por tanto, uno de ellos se comporta como Memoria Interna, mientras que el otro funciona como Bloque de Registros.

Para evitar confusiones, supondremos que sólo hay seis **Registros de Trabajo**,

B, C, D, E, H y L,

e ignoraremos al Segundo Grupo:

B', C', D', E', H', y L'.

Otra particularidad de estos registros es que, además de ser registros generales de 8 bits, se pueden agrupar en pares de **Registros Apareados**: **BC, DE y HL**. Así pues, los contenidos de B y C, por ejemplo, pueden contener datos de 8 bits, que unidos formen una dirección de memoria de 16 bits o un dato de 16 bits. Gracias a ello, los seis registros de este grupo pueden usarse tanto para almacenar datos de 8 bits como para guardar apuntadores de 16 bits para el direccionamiento de la memoria.

- Otro grupo de registros, presentes en todos los microprocesadores, son los **Registros de Direcciones “Puros”**. En el Z80 hay, como en todo microprocesador:
 - Un **Contador de Programa, PC**, que contiene la dirección de la instrucción que ha de ejecutarse a continuación
 - Un **Puntero de Pila, SP**, que señala la parte superior de la pila en la memoria. (En el Z80, señala la última entrada real en la pila. Cabe mencionar además, que la pila crece “hacia abajo”, es decir, hacia direcciones de memoria más bajas).
- En el Z80 existen también unos registros especiales, denominados **Registros Índices** que son: **IX** e **IY**. Su funcionamiento y direccionamiento es muy particular, y bastante complejo. Nos bastará con conocer su existencia y saber que se trata de registros de **16 bits** que se utilizan para las **instrucciones indexadas** (para la indexación en general).
- Para finalizar este apartado, nos queda ocuparnos de un registro fundamental en todo microprocesador: **El Acumulador**. En el Z80 tenemos:
 - **A** el **Acumulador Principal**,
 - **A'** el **Acumulador Secundario** (normalmente inactivo).
- Junto al acumulador aparece un **Registro de Estado** denominado **F** (y también tenemos el **F'**), en el que se encuentran las famosas “**banderas**”. (Este registro de estado, **no es directamente modificable por el usuario**, sino que va siendo modificado por el propio programa debido a las situaciones “especiales” que en el programa se den). La función de las banderas del registro F se explicará más adelante.

Nota: Para simplificar las cosas supondremos siempre, salvo que se diga lo contrario, que sólo disponemos de los registros: **B, C, D, E, H, L, A** y **SP**.

B) LA PILA:

Una Pila es lo que formalmente se llama una estructura de datos del tipo **LIFO** (**Last In First Out**, es decir, el **Último** en **Entrar** el **Primero** en **Salir**). Está formada por un conjunto de posiciones de memoria adscritos a dicha estructura de datos. Su característica esencial es que se trata de una estructura cronológica: el primero de los elementos introducidos en la pila ocupa siempre el fondo de la misma, mientras que la introducción más reciente está siempre en su parte superior. Su organización es comparable a la del portabandejas de la barra de una hamburguesería: las bandejas se apilan sobre una base que se hunde en el pozo de la barra conforme aumenta el peso, se colocan y se cogen por arriba, de manera que las últimas en llegar al montón son siempre las primeras en salir de él.

La pila en el Z80 sólo es accesible mediante 2 instrucciones:

- **Push:** Guardar una dirección o dato de 16 bits en la dirección apuntada por el puntero SP (Puntero de la Pila, en inglés **Stack Pointer**)
- **Pop:** Extraer una dirección o dato de 16 bits de la dirección apuntada por el puntero SP.

En definitiva, lo que hacen respectivamente cada una de estas instrucciones respectivamente es: Guardar un elemento en la pila y Extraer un elemento de la pila

El Puntero de la Pila (SP) se incrementa o decrementa automáticamente tras ejecutar estas dos instrucciones, de modo que siempre ha de estar apuntando a la última entrada realizada.

Existen también otras instrucciones para extraer elementos de la pila (que de momento no nos interesan). La pila es necesaria para aplicar tres recursos de programación:

- Subrutinas.
- Interrupciones.
- **Almacenamiento Temporal de Datos** (que es lo que a nosotros nos interesa!!!).

La PILA, se crea en el Z80 de la siguiente forma: dentro del MicroProcesador se reserva un Registro (el SP), para almacenar el puntero de la pila, es decir, la dirección de su elemento superior (o a veces, la dirección de su elemento superior mas uno). A continuación se crea la pila como una zona de la memoria; de esta forma bastan los 16 bits que ocupa el puntero para señalar cualquier posición de la pila.

El interés que para nosotros tiene la pila es que puede ser fácilmente “relocalizada”, es decir, que su posición dentro de la memoria puede cambiar con sólo cambiar la dirección contenida en el puntero de la pila (SP). Esta relocalización la llevan a cabo casi siempre

todos los programas comerciales y es fundamental tenerla en cuenta a la hora de desprotegerlos (porque el BASIC, como todo programa, tiene su propia pila alojada en unas direcciones de memoria fijas, a partir de **&HF000**, y si alteramos la pila desde CM y luego queremos volver al BASIC, el ordenador se quedará bloqueado!!!). ☹

C) LAS BANDERAS o REGISTROS DE ESTADO (F):

El Registro de Estado es un registro de **8 bits**, cuya función es la de “llevar la cuenta” de las situaciones excepcionales que se dan en el interior del microprocesador. El contenido del Registro de Estado puede verificarse mediante instrucciones especiales o leerse mediante otras instrucciones. Las instrucciones condicionales provocan la ejecución de un nuevo programa. (o el salto a otra parte del programa) en función del valor de uno de los bits del Registro de Estado. Los bits del referido Registro de Estado reciben los nombres de:

S, Z, H, P/V, N y C

(Ojo, no confundir la Bandera H y C con los Registros del mismo nombre!!!!).

Las Banderas, que son cada uno de los Bits del Registro de Estado F, que más vamos a utilizar y que por lo tanto más nos interesan son dos: **C** y **Z**.

1. **Acarreo (C)** : El Bit de Acarreo o Bandera de Acarreo desempeña una función doble:

- en primer lugar indica si una operación de suma o resta ha dado lugar a un acarreo (llevarse uno).
- en segundo lugar, es el noveno bit de las instrucciones de desplazamiento y rotación.

Son muchas las instrucciones que modifican el estado de este bit (**0** ó **1**), pero en general se puede decir que todas las operaciones aritméticas lo igualan a **0** ó **1**.

2. **Cero (Z)**: El Bit de Cero o Bandera de Cero se utiliza para verificar si es cero el valor de un byte que se ha calculado o que se está transfiriendo. También se usa en instrucciones de comparación, para señalar la coincidencia, y en algunas otras funciones. Si se produce una operación con resultado 0 o si se transfiere un dato y el byte vale 0, el bit **Z** se fijará a **1**; en caso contrario se fijará al valor 0. (También desempeña otras muchas funciones.....):

5. INSTRUCCIONES DEL Z80.

Las instrucciones que vamos a examinar son las que nos serán necesarias en un futuro (sólo veremos las **IMPRESINDIBLES!!!**). Son las siguientes:

- **DI:** Desconecta las interrupciones automáticas del Lenguaje Máquina. Debéis utilizarla siempre antes de conmutar páginas de la memoria o Slots (es recomendable). Una explicación medianamente “buena” de esta instrucción ocuparía unos 20 ó 30 folios...
- **PUSH XX:** Introduce el par de registros XX en la pila. XX puede ser: SP, BC, DE, HL, AF, IX e IY. El contenido de cualquiera de estos registros se guarda en la pila y el puntero de la pila (SP) se decrementa en dos direcciones (2 bytes). Olvidad la explicación “técnica” y recordad que esta instrucción sirve para GUARDAR DATOS EN LA PILA. Como el número de registros en CM es muy limitado es necesaria una zona de la memoria (la PILA), en la que se guarden temporalmente diversos datos y direcciones, ya que los registros están siempre ocupados (NO pueden utilizarse los registros como si de simples variables Basic se tratara!!!) ☹
- **POP XX:** Extrae de la pila el par de registros XX. El contenido de la posición de memoria direccionada por el puntero de la pila (SP) se carga en el par de registros XX y a continuación se incrementa en dos bytes dicho puntero. Esta instrucción hace exactamente lo contrario que PUSH, es decir, nos devuelve el valor que habíamos guardado en la pila en el par de registros XX. Nos servirá para “SACAR” VALORES PREVIAMENTE GUARDADOS EN LA PILA.
- **INC X o XX o:** Incrementa en una unidad el contenido del registro X (que puede ser A, B, C, D, E, H y L) o del par de registros XX. Imaginemos el siguiente ejemplo:

1. HL = 255	LD HL, 255
2. HL = HL + 1	INC HL
3. HL = 256	(HL = 256)

- **DEC X o XX o ...:** Decremento el contenido del registro X o de el par de registros XX. Hace exactamente lo contrario que INC como habréis notado... ☹

XX ← XX – 1

- **JP DIR o CC, DIR o ...:**
 - JP DIR salta a la posición de memoria especificada por DIR, es decir, la ejecución del programa “salta” a dicha posición de memoria (viene a ser como un GOTO, para que nos entendamos...).

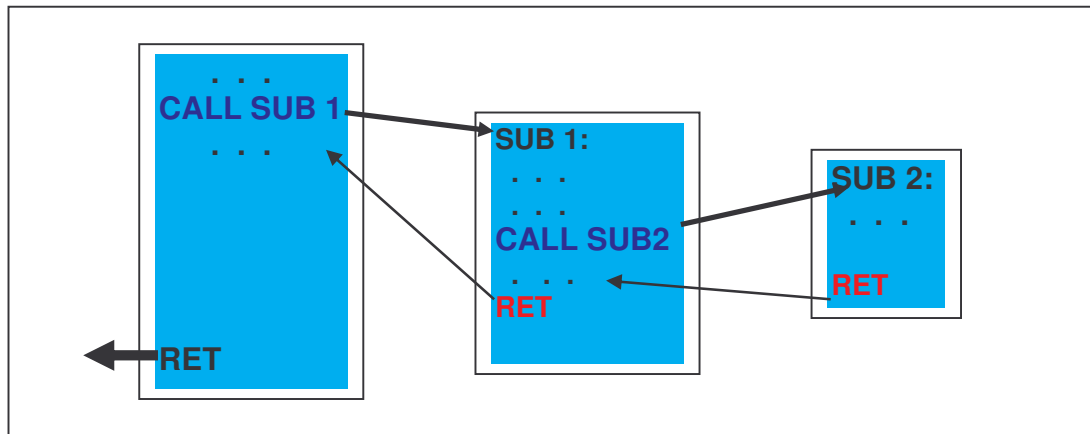
- JP CC, DIR salta, transfiere la ejecución del programa, a la posición de memoria especificada por DIR sí y sólo si se satisface previamente la condición CC. Sólo si se produce la condición se da el salto. Las condiciones que pueden verificarse para realizar los saltos son las conocidas banderas (C, Z, S, P/V, etc...). Ved el siguiente ejemplo:

1. LD BD, 2	B = 2
2. DEC B	B = B - 1 : B = 1
3. JP Z, BASIC	Ir al BASIC si B = 0 . (NO se cumple!!!)
4. DEC B	B = B - 1: B = 0
5. JP Z, BASIC	Ir al BASIC si B = 0. (Se cumple.)

A la izquierda tenéis el Código Real, y a la derecha los comentarios a éste complejísimo programa... ☺

- **CALL XX o CC, XX o:** La ejecución del programa salta a la dirección de memoria XX (o bien salta a la dirección de memoria XX si se satisface la condición CC). La ejecución del programa continúa por la nueva dirección XX hasta que se encuentra la instrucción **RET** o **RET CC** momento en el cual, la ejecución retorna a la dirección de memoria posterior a la que contenía la instrucción CALL. ¿Lo habéis leído bien....?. ¿Si...?. ¡Pues ya lo estáis olvidando...!. Esta instrucción viene a funcionar de modo similar a GOSUB y RETURN del BASIC (¡mentira podrida!, pero como no quiero explicaros a fondo como funciona, otras 20 hojitas de nada, ¡con eso vais de pique!.....).
- **RET o RET CC:** Retorno de una Subrutina, o retorno condicional de una Subrutina. Para volver al BASIC desde el CM debéis utilizar al final de vuestros programas esta instrucción, cuyo “formato” o byte que la representa es **&HC9** (ó **201** en decimal).

Fijaos en el esquema y entenderéis, o eso espero, su funcionamiento básico:

Mecanismo de la Subrutina:

- **IN r, (C):** Carga el registro **r** (A, B, C, D, E, H y L) a partir del puerto **C**. Se lee el dispositivo periférico direccionado por el contenido del registro **C** y el resultado se carga en el registro especificado. Consultad la instrucción **IN** de vuestro manual BASIC (que es casi idéntica!!!!). Consultad también de vuestro manual la asignación de puertos de entrada y salida (al final del manual, si es que viene...).

Ejemplo:**IN B, (&HA8)**

B = contenido del puerto &HA8, es decir, la asignación de las páginas de memoria (Slots).

- **OUT (C), r:** Si **IN** sirve para leer el contenido del puerto **C**, ¿**OUT** servirá para...?. ¡Premio!, para enviar un dato a dicho puerto. Técnicamente se dice que sirve para dar salida al contenido del registro **r** a través del puerto **C**: el contenido del registro de **8 bits** especificado se lleva al dispositivo periférico direccionado por el contenido del registro **C**. El registro **r** puede ser: A, B, C, D, E, H y L.

Consultad de nuevo en vuestro manual de BASIC: la función **OUT** del BASIC y la del CM son casi idénticas. A nosotros esta instrucción nos va a servir para activar la configuración de memoria TODO RAM y la configuración ROM/RAM.

- **LD destino, origen:** El contenido de la dirección, registro o dato de origen se carga en la dirección o registro de destino. Fijáos en los ejemplos (hay multitud de instrucciones “variantes” de LD).

1. **LD A, 201** A = 201
2. **LD A, B** A se carga con el contenido de B. (B no se modifica).
3. **LD HL, 500** HL = 500
4. **LD HL, (DIR)** HL = contenido de la dirección de memoria DIR.
5. **LD (DIR), HL** En la dirección de memoria DIR se introduce el valor de HL.
6. **LD A, (DIR)** A = contenido de la dirección de memoria DIR.
7. **LD (DIR), A** En la dirección de memoria DIR se introduce el valor contenido en A.

Y podría seguir, y segur..., y ahora no llenaría 20 hojas, ¡tendría que escribir más de 100!. De **forma genérica** se puede resumir:

LD DD, (NN)	LD R, N
LD (NN), DD	LD R, R
LD DD, NN	LD (BC), A
LD NN, DD	LD (DE), A
LD (HL), n	LD (HL), A
....	...

Donde **DD** sería un registro de **16 bits** (HL, DE, BC...), **NN** sería una dirección de memoria o un dato de **16 bits** (de 0 a 65.535 en ambos casos), **R** sería un registro de **8 bits** (A, B, C, D, E, H y L) y **N** sería un dato numérico de **8 bits**, un byte de 0 a 255...

Esta instrucción es **fundamental en CM**, y creo que es la que más se utiliza, ya que es la más abundante (hay más de 50 LDs distintos). Más adelante la trataremos en diversos ejemplos.

Tras haber leído todo esto estoy seguro de que tenéis un cacao mental que es demasiado... Código Máquina, lo que se dice Código Máquina no tenéis ni pajoletara idea, ¡pero os ha quedado un dolor de cabeza que da gusto...! (“No hay mal que por bien no venga...”, je, je, je...)

6. LOS EJERCICIOS.

Dada la importancia de este tema, y su CONSABIDA DIFICULTAD, convendría que realizaseis los siguientes ejercicios (una vez que hayáis leído y comprendido todo lo anterior, eh!!!). En esta ocasión los ejercicios son MUY IMPORTANTES (como importantísimo sería que cogierais un ensamblador, Gen, Rsc/Rsc2, asMSX ☺, y fueseis practicando “en vivo”; o también que le echaseis un vistazo al libro CM que tenéis sujetando la pata coja de la cama, por ejemplo... ☺)

1. ¿A qué tipo de programa corresponde este LEADER y cual es el nombre del programa?.

DO DO DO DO DO DO DO DO DO DO DO 53 4F 46 54

2. ¿A qué tipo de programa corresponde el siguiente LEADER y cual es el nombre del programa?. (Fíjate bien!!!)

EA EA EA EA EA EA EA EA EA EA EB 00 00 00 00 00 00

3. En vistas del siguiente LEADER: ¿Cuáles son las direcciones de Inicio, Fin y Ejecución del programa?

00 88 00 C8 00 88 (nota: todos los números están en hexadecimal!!!)

4. ¿Es posible que un programa con un nombre de más de 6 caracteres cargase desde el BASIC (con CLOAD, LOAD O BLOAD)?.
5. ¿Cuáles son los **Registros de Propósito General** que posee el procesador del MSX?
¿Cuáles son los **Registros Apareados** que se pueden formar?.
¿Es el Registro **IX** un registro de **8 bits**? Y el registro **A**, ¿es de 8 ó de 16 bits?.
6. ¿Puede un **registro de 16 bits** cargarse con un **dato de 8 bits**?.
¿Puede un **registro de 8 bits** cargarse con un **dato de 16 bits**?.
7. ¿Podré introducir el valor **67000** en el registro **HL**? ¿Y el valor **35500** en el registro **A**? ¿Y en el **B**?.
8. ¿Qué sucederá si cargo en el **Puntero de la Pila** el valor **45000** (la dirección 45000 para ser más exactos? (Esta pregunta tiene mucha miga...!!!)
9. ¿Qué significa **LIFO**? ¿Qué es una **PILA**? (las de corriente no valen!!!) ☹

10. Haz lo siguiente:

- B = 245
- C = 28
- A = 25000
- A = (contenido de la dirección 34000)
- HL = 4500
- DE = 25
- A = (contenido de HL)
- HL = contenido de la dirección DIR
- Carga en la dirección genérica DIR el valor 500
- Carga en DIR el valor de BC

11. Haz lo siguiente:

- B = 77
- Réstale 4
- Comprueba si es 0 y si no lo es salta a BASIC

12. Haz lo siguiente:

- B = 201
- C = B
- Guarda BC en la PILA
- Lee el contenido del puerto &HA8 (asignación de la memoria), y guardalo en el registro E.
- Extrae BC de la PILA.
- Envía el contenido de B y C al puerto &H99.

13. Si el registro **B** contiene **255** y lo incrementamos en una unidad, ¿qué sucederá con el valor contenido en ese registro?. (esto nos es fácil de responder..., y os adelanto que la respuesta NO ES QUE EL ORDENADOR SE BLOQUEA!!!)

N.B. Cuando digo “*haz lo siguiente*”, me refiero a que debéis hacer un pequeño programa en CM que realice las operaciones que os indico. Por ejemplo, si os digo que hagáis:

- $E = 4$
- Incrementar E
- $D = 248$
- Decrementar D

Deberíais hacer lo siguiente:

- LD E, 4 $E = 4$
- INC E $E = E + 1 ; E = 5$
- LD D, 248 $D = 248$
- DEC D $D = D - 1 ; D = 247$

(Lo que va en “azul” son las instrucciones CM, y lo de “negro” los “comentarios” de lo que hacen dichas instrucciones!!!) ☺

P.D. En primer lugar, quiero realizar una aclaración de suma importancia... Este curso fue creado originalmente por **FRANCISCO FUENTES LORENZO**. El formato original del mismo son fotocopias escritas a máquina, y que yo **JOSÉ MANUEL SOTO**, he intentado pasar a pdf para luego aplicarle soft de OCR, pero viendo los desastrosos resultados he optado por teclearlo todo letrita a letrita en formato WORD, respetando fielmente el original....

Para cualquier aclaración, comentario, sugerencia, etc, etc... no dudéis en escribirme. Me gustaría que todo aquel que tenga conocimientos sobre el tema, y pueda contribuir a mejorar este curso, ampliarlo, corregirlo, etc, que se anime y me escriba. También me gustaría me escribierais para darme vuestra opinión sobre el mismo, si lo habéis leído, y tenéis intención de seguirlo, etc, etc, etc... Y aprovecho, aún a riesgo de ser pesado que por favor, todos los que tengáis material MSX por ahí que sea interesante para el resto (libros, documentación técnica, etc, etc, ...) o si sabéis de alguien que lo tenga, que lo escaneéis y lo paséis a pdf para compartirlo con el resto y que todo ese valioso material no desaparezca con el paso del tiempo!!!).

Gracias al Paco y al Robsy por animarme a llevar a cabo esta iniciativa, y a Karloch por darle difusión. Y gracias, mil veces gracias al Paco que casualidades de la vida nos hemos vuelto a “encontrar” después de mucho tiempo, mucho, mucho tiempo... Gracias y mil veces gracias Paco por tus cartas, los juegos que gracias a ti conseguí, las dudas que me resolviste, tu infinita paciencia y por haber escrito este curso en su día “exclusivamente para mi”... (ja, ja, ja... quien te iba a decir esto por aquel entonces, ¿eh?).

Va por ti, MAESTRO...!!! ☺ ☺ ☺

jose_manuel@mixmail.com

CURSO DE DESPROTECCIÓN **DE PROGRAMAS**

CAPITULO 3

1. INTRODUCCIÓN.

Una primera aclaración o consejo antes de empezar: dado el tiempo transcurrido desde que leísteis este “culebrón galáctico”, no estaría mal que lo releýeseis de nuevo, sobre todo el primer capitulo de la saga, y también que repasaseis las malditas SOLUCIONES a los ejercicios del capitulo segundo.

Al plantearme la realización de la tercera entrega de este emocionante historia, una profunda duda atenazaba mi pluma... y no, ¡no!, no era la falta de tinta... La cuestión es si debía realizar un capitulo centrado en la teoría, como el segundo, o encaminado a la práctica, como en cierta forma el primero...???. Entendámonos, tenía claro la inclusión de bastantes ejemplos, imprescindibles para una buena comprensión de las ideas a ilustrar, pero no tenía tan claro el enfoque general que debía utilizar: ¿teórico con ejemplos, puramente práctico..., o alguna posición intermedia...?. Al final, se impuso el enfoque pragmático, es decir, el enseñar mediante la práctica, práctica y práctica directa, aunque introduciendo un poquito de teoría...

2. “PAPÁ” MICROSOFT, “MAMA” KONAMI... ¡y EL “TIO” GRENLIM....!

Imagino la cara de mi amigo Antonio al leer esto y supongo que, una vez más, correrá a recordarme mi exagerada e injustificada – según él -, admiración por Grenlim Graphics. Tampoco escatimará esfuerzos en advertirme de lo exagerado que resulta ponerla junto a estos dos auténticos monstruos de la programación. En mi defensa diré que la graduación expuesta (Microsoft-Konami-Grenlim) responde a una exacta jerarquía en lo que a mis gustos se refiere, basada en el nivel de programación sobre el MSX que, a mi juicio, alcanzaron en su momento. Ciertamente, que otras empresas han superado con creces a Grenlim programando, pero el hecho de que el primer videojuego que compré fuese suyo, ¡y fuese muy bueno! (fue el Avenger), es algo que impactó como una bala de Magnum 45 en mi mente infantil. Un recuerdo imborrable, ya lo creo... Pero dejemos las bonitas palabras y empecemos de una maldita vez...

El presente capítulo estará dedicado a la **DESPROTECCIÓN**, entendido el término como lo definí en el capítulo segundo, de programas “TURBO” (programas que hacen rayitas multicolores en pantalla al cargar) como son los de GRENLIM, TOPO, y muchos otros...

A) TURBO GRENLIM 16 VÁLVULAS G.T.I.

Coge el **LISTADO 1** de los que incluyo como **ANEXO** de este capítulo. Es el *Analizador de Slots de Grenlim*. Échale un breve pero intenso vistazo y haz con el lo siguiente:

- Tecléalo en el GEN, el RSC/2, asMSX ☺..., ensámblalo y una vez sin errores grábalo para futuras utilizaciones.
(Si quieres ahorrarte este trabajo -¡gandul!-, extraelo de algún programa original).

No es necesario saber como funciona, con tal de que funcione (enfoque pragmático, recuerda). De cualquier forma si te pica la curiosidad al respecto házmelo saber...

- A continuación, coge el *Loader “Estándar” de Grenlim*, **LISTADO 2**. (Lo de estándar entrecomillado viene a cuento porque esta empresa, en sus primeros tiempos, no utilizaba este programa cargador en sus juegos.

Supongo que el listado ensamblador te sonará a chino, como mínimo... ¡No te preocupes, a mí me ocurre igual...!, y como lo mío no son los idiomas, no voy a malgastar tu tiempo y el mío en intentar explicarte instrucción por instrucción el funcionamiento del programita.

Por razones que no vienen a cuento, y que comprenderás en cuanto profundices en tu estudio del librito sobre el Z80 de Rodnay Zaks, los programas cargadores de “turbo” utilizan TODOS (los que conozco y he visto hasta hoy) la siguiente convención:

a) El registro **IX** se usa para cargar la dirección inicial de almacenamiento en memoria del bloque a cargar.

b) El registro **DE** contiene la longitud en bytes del bloque de datos a cargar.

c) El acumulador (**A**) suele cargarse con el valor **255** o cercano, y la bandera de acarreo (**C**) suele ponerse a **1** antes de....

d) Llamar a la **SUBROUTINA GENERAL DE CARGA**, que existe SIEMPRE, y cuya misión será cargar los bloques desde el casete (en función de los datos contenidos en **IX, DE, A** y **C**).

El requisito **c)** no es tan genérico como los otros tres.

Como toda convención matemática, ésta puede parecer un tanto arbitraria. ¡Nada más falso!; he usado alegremente la palabra “convención” para dar a entender un uso continuado o generalizado, pero en realidad no se trata de convención en sentido estricto (no es que los programadores se hayan reunido para decidir que siempre deba ser así, ¿está claro?). Nada impide cargar programas turbo de otra manera (piensa por ejemplo, en los que hacen rayitas de un mismo color al cargar, a los que no puede aplicarse esto)....

La primera diferencia que salta a la vista entre el método de cargar programas BASIC (bload) y éste, es que no existe dirección inicial ni dirección de ejecución. Si lo piensas bien comprenderás que no son necesarias, pues con una dirección inicial y una longitud de bloque puede cargarse cualquier fichero secuencial desde cinta.

Pero volvamos al programa cargador de Grenlim:

Se inicia en la dirección ram **&HD800** (55.296) y acaba en **&HD980** (55.680), aunque realmente acabe en **&HD8F1** (55.537). Vamos a comentarlo paso a paso mientras sea necesario:

LD A, 2
CALL &H5F

Estas dos instrucciones activan el Screen 2, en este caso. Esto se consigue cargando en el acumulador el número de screen deseado y llamando a la rutina BIOS situada en la dirección **&H5F**, que se encarga de activarlo.

DI

Desactiva las interrupciones del Lenguaje Máquina.

LD IX, &HD8ED	IX= Direcc. Inicio Ram almacenamiento.
LD DE, &H4	DE= Longitud bloque (bytes)
LD A, &HFE	
SCF	
CALL &HD837	

El registro **IX** se carga con la dirección RAM **&HD8ED** (dirección de inicio en memoria de la “minicabecera” grenlim). En el registro **DE** se introduce el valor **4**, que es el número de datos a cargar desde el casete (longitud de la “minicabecera”). En (**A**) se carga el valor **254** y el acarreo (**C**) se pone a uno con **SCF**. A continuación se realiza

una llamada –**CALL**– a la **SUBROUTINA GENERAL DE CARGA**, que introducirá en memoria (a partir de **&HD8ED**) cuatro datos (**DE**) desde el casete.

Hagamos un pequeño alto en el camino para explicar tres cosas de suma importancia:

- Al ir a desproteger programas de este tipo por vuestra cuenta y estar desensamblando el programa cargador, deberéis buscar SIEMPRE una secuencia de instrucciones lo más parecida posible a la última comentada. Huelga decir que cuanto más similar sea la encontrada a ésta, tanto más fácil será desproteger ese programa.... ¡y viceversa! (cuantas más instrucciones existan antes de la secuencia o intercaladas con ella más difícil será la desprotección).
- También es evidente, pero no por ello dejaré de decirlo; cuantas menos instrucciones encontremos desde la dirección de ejecución del programa cargador (inicio **REAL** del mismo) hasta dar con la secuencia mencionada, tanto más fácil será, por lo general, desproteger ese programa. Es obvio si pensamos que las instrucciones precedentes pueden modificar el estado de los slots, la peligrosa pila, etc, con lo cual al retornar al Basic tras cargar un bloque de datos deberemos revertir a su estado original nuestro MSX (lo que requiere amplios conocimientos por nuestra parte).

Como puedes observar en el listado ensamblador, los programas de Grenlim lo único que hacen antes de cargar un bloque de datos es activar el Screen 2, cosa que a primera vista no parece muy grave, ¿verdad?.

- La peculiaridad genial que distingue a Grenlim de las demás empresas que han usado el sistema “turbo” para cargar sus programas es, precisamente, la secuencia de instrucciones analizada y lo que ésta significa. Así, mientras las demás empresas tenían que realizar un programa cargador para cada juego, pues los contenidos de **IX** y **DE** variarían para cada bloque de datos, los chicos de Grenlim, más pillos y gandules que otros, se ahorraron este tedioso trabajo de tecleado mediante una sencilla pero genial variación.....¡INCLUYERON UNA MINICABECERA!, precediendo a cada bloque que contenía la dirección de inicio (**IX**) y la longitud (**DE**) del bloque tras ellas. Son esos 4 bytes que veíamos al explicar las instrucciones.

Hicieron posible así un **CARGADOR ESTANDAR PARA TODOS SUS PROGRAMAS**, ya que las direcciones están grabadas en cinta y no es preciso reescribir el programa cargador para cada nuevo juego.

¡Pues vaya estupidez, pensarás...!. Pues vaya estupidez la de los demás, que NO lo hicieron; malgastando tiempo de programación, léase “dinero”, tontamente.

En honor a la verdad debo decir que Grenlim siguió, consciente o inconscientemente, los pasos de "PAPA" MICROSOFT, el primero en hacerlo para MSX con sus blood's. Por algo está MICROSOFT la primera de mi lista....

Hecho este pequeño paréntesis, continuemos:

JR NC, &HD805

Si el acarreo es uno (1) sigue buscando un bloque de datos en la cinta para cargarlo, porque no se ha cargado todavía nada. Si el acarreo (C) vale cero (0) continua.

LD IX, (&HD8ED)

LD DE, (&HD8EF)

LD A, &HFF

SCF

CALL &HD837

JR NC, &HD805

Si se ha cargado la cabecera, pues sino no se ejecutarán estas instrucciones, **IX** se carga con el valor contenido en la dirección de memoria **&HD8ED** (donde se guardan los dos primeros bytes de la minicabecera, o sea, la DIRECCIÓN DE INICIO del bloque de datos grabado a continuación en la cinta). El registro **DE** se carga con el valor contenido en **&HD8EF**, donde están almacenados los dos últimos bytes de la minicabecera, es decir, la LONGITUD del bloque a cargar). Se actualizan el acumulador (**A=255**) y el flag de acarreo (**C=1**) para llamar a la SUBROUTINA GENERAL DE CARGA, que comienza en la dirección **&HD837**.

Si deseas saber como funciona en concreto dicha SubRutina no tienes más que preguntar. Como se trata de un tema harto largo y complejo no voy a extenderme en su explicación, pero si te diré que esta SubRutina carga el bloque de datos desde el casete si previamente a llamarla con call se han inicializado los registros **IX**, **DE**, **A** y el flag **C** con los valores necesarios.

Ya no necesitas seguir comentando el programa cargador, puesto que una vez ejecutadas estas instrucciones tendremos cargado en memoria el bloque que empieza en la dirección de memoria contenida en **&HD8ED** y cuya longitud está contenida en **&HD8EF**. Ahora bien, ¿Cómo hacernos con el control del sistema y evitar que el programa cargador siga ejecutándose (e.d., cargando bloques hasta el final, y haciendo luego funcionar el juego cargado)?. Pues, en el caso de Grenlim (no modifica slots, ni la pila ni nada) esto es de lo más sencillo.

Cambiamos:

LD HL, (&HD8ED) (2A, ED, D8) 3 bytes

Por:

RET (C9)
NOP (0) 3 bytes
NOP (0)

Y una vez cargado el bloque, el propio programa cargador retornará al BASIC devolviéndonos gentilmente el control del sistema.

Ahora es el momento de que cargues el LOADER de Grenlim con el MON (o RSC II) e introduces la mencionada variación. A continuación grábalo con **BSAVE"nombre", &HD800, &HD980, &HD800** (o GB). Ya tienes un copión estándar para Grenlim en ciernes; te falta tan sólo un pequeño módulo base en BASIC que controle la E/S desde el teclado y cargue el "TURBO" para grabar BASIC. Este programilla (**Listado 3**) lo único que hace es cargar el programa cargador modificado – valga la rebuganancia- y ejecutarlo, con **BLOAD"CAS:"**, R primero y **DEFUSR=&HD800:A=USR(0)** después. Este programa nos carga los bloques "turbo" desde la cinta (ejecutando el programa cargador) y nos devuelve el control al módulo base en BASIC. Entonces nosotros calculamos la Dir. Final del Bloque (**DF = Dir contenida en &HD8ED + Dato contenido en &HD8EF**) y lo grabamos en Basic con:

BSAVE"nombre", (&HD8ED), Dir. Final, (&HD8ED)

Los paréntesis significan "contenido de la dirección de memoria X". ¡Por cierto!, si quieres leer ese contenido directamente puedes hacer:

?PEEK (&HD8ED)+256*PEEK(&HD8EE) lo que te dará la dir. Inicial del bloque.

?PEEK(&HD8EF)+256*PEEK(&HD8F0) lo que te dará la long. del bloque a cargar.

TE habrás dado cuenta de que los bloques de datos de Grenlim están ESTRUCTURADOS, e.d., su dirección de ejecución y su dirección de inicio SIEMPRE COINCIDEN. Esto no tiene porque ocurrir, y de hecho no ocurre con otras marcas.

Si repites la operación de carga/grabación con todos los bloques del programa, ya "casi" lo tendrás desprotegido. El problema consiste ahora en ingeniárselas para cargarlo desprotegido (¡y funcionando a ser posible!). Esta cuestión será una autentica comedia de coco con otros programas, pero no con los de Grenlim. Además de estar ESTRUCTURADOS, éstos también se AUTOEJECUTAN (los bloques), e.d.; se transfieren automáticamente a las direcciones de memoria definitivas en las que residirán los datos cargados. Por todo ello, para cargar un juego desprotegido de esta empresa, el único cargador que precisas es el siguiente:

10 rem LOADER GRENLIM
20 COLOR 15, 1, 1: SCREEN 2
30 BLOAD"CAS:", R: GOTO 30

Teclea el anterior, y complejo, programa y grábalo en cinta. Tras él, graba el **Analizador de Slots de Grenlim**, seguido de los **bloques del programa desprotegidos** (y en el mismo orden del original)....¡et voila!. Todo listo para la carga.

Desconozco si con unidad de discos los juegos de Grenlim plantean problemas específicos de carga, pero lo que es con el casete, ¡la cosa va como la seda!. Ahora bien, si grabas el programa desprotegido en disco no podrás percibir un pequeño-GRAN detalle; **el programita desprotegido tarda ¡MUCHO MÁS! en cargar que el original (o “turbo”)**. Evitar el pirateo, ja, ja, ja.... y agilizar la carga, fueron las dos poderosas razones por las que los programadores idearon este sistema especial de grabar/cargar datos en/con el casete.

3. NIVEL 2: “LEARNING TO FLY”.

Puede decirse que hasta el momento no hemos progresado gran cosa. ¡Cierto!, hemos desprotegido programas de Grenlim, pero esto, gracias a la propia Grenlim, ha resultado tan fácil que incluso nos ha dejado con mal sabor de boca. Va siendo hora de atreverse con algo bastante más difícil; p.ej. el juego **BLACK BEARD**, de **Topo Soft**.

Para empezar, hazte con una copia protegida del mismo y procede como sigue:

1. Carga la presentación, el nombrecito saltarán, estando muy atento para hacer CTRL+STOP nada más ejecutar ésta.
2. Teclead **LOAD”CAS:”** Una vez hayas cargado el pequeño programita basic, examínalo. Verás que a continuación hay grabados dos bloques, llamados TEST y LOAD. El primero es el analizador de Slots de Topo, el segundo es el Loader o programa cargador (os envío su listado aparte para que lo veáis). Extraed los dos programas en una cinta aparte.

Lo primero que debes hacer para desproteger un programa es desensamblar (MON o RSC II) su programa cargador, y sacarlo por impresora si puedes. En todo caso puedes copiarlo en papel, a mano, o a máquina...

A continuación averigua CUANTAS cargas, léase bloques, tiene ese programa. Para hacerlo debes localizar la “secuencia” antes mencionada partiendo, en tu búsqueda, desde la dirección de ejecución del programa cargador (no desde la inicial, ¡¡mucho ojo!!). Puede ser útil cargar el programa normalmente antes de empezar a desproteger, contando con los dedos los bloques que tiene.....

En el caso concreto del **BLACK BEARD** hallarás estas “secuencias” de instrucciones:

LD IX, &H9C40 Primer bloque.
LD DE, &H1BBA Screen

.....
LD IX, &H2328 Segundo bloque
LD DE, &H32C8

.....
LD IX, &H55F0 Tercer bloque
LD DE, &H2A00

.....
LD IX, &H8400 Cuarto bloque
LD DE, &H4FFF

Visto esto, no es necesario ser Einstein ni tener el C.I. de Goethe para darse cuenta de que el juego tiene **cuatro bloques protegidos**. En este momento conviene hacer una pausa y recapacitar, ordenando de paso la información que conocemos. Yo lo solía hacer así: (es muy útil, casi vital)

BLACK BEARD: Loader → **D6D8 –D81C – D6D8**

1er. Bloque → Ini = **&H9C40** Longitud = **&H1BBA** (screen)
2do. Bloque → Ini = **&H2328** Longitud = **&H32C8**
3er. Bloque → Ini = **&H55F0** Longitud = **&H2A00**
4to. Bloque → Ini = **&H8400** Longitud = **&H4FFF**

(&HD6E6) – Dir. INICIO.
(&HD6E9) – LONGITUD bloque.

>&HD71F – SUBROUTINA GENERAL DE CARGA.

Modifiquemos ahora el programa cargador, con el objeto de que cargue un bloque “turbo” y nos devuelva el control al Basic. Coloquemos un **RET** y dos **NOP** tras la primera llamada a la Subrutina General de Carga, es decir, en **&HD6F1**.

Fíjate en el principio del programa cargador. Verás que, antes de cargar nada, se hacen una serie de llamadas a subrutinas; subrutinas de las que no conocemos los efectos. Luego, en principio, es peligroso ejecutarlas (en este caso estas rutinas tienen por objeto analizar

los slots y subslots y conectar la configuración de memoria TODO RAM). No debemos ejecutarlas pues de hacerlo nuestro MSX se quedaría colgado al intentar retornar a un Basic desconectado. Para saltarnos estas llamadas deberemos POKEAR las direcciones de memoria justamente anteriores a **LD IX, &H9C40**, e.d., desde **&HD6E1** hasta **&HD6E3**. Introduciremos los valores **0, 0** y **F3** (o sea, **NOP, NOP** y **Di**). En realidad basta con pokear una dirección antes de la citada instrucción, pero yo prefiero ser un poco más cauteloso.

Estamos en condiciones de cargar el primer bloque del programa; cosa que haremos con **DEFUSR=&HD6E1:A=USR(0)**. ¡Por cierto!, no estaría de más que antes grabases el programa cargador modificado en una cinta aparte por si se te quedase bloqueado el ordenador. Así te ahorrarías tener que repetir todo el proceso (¡créeme, sería muy raro que no se te colgase a menudo!).

Carga el primer bloque, la pantalla de presentación, y grábalo con **BSAVE"nombre", &H9C40, &H9C40+&H1BBA (&HB7FA)**. Puedes ejecutar la portada para verla si lo deseas, con:

10 SCREEN 2: DEFUSR=&H9C40:A=USR(0)

Vamos a por el siguiente bloque. Pero ¡OJO!, antes de cargarlo hay que actualizar la dirección de inicio y la longitud, haciendo:

POKE &HD6E6, &H28 La dirección se debe pokear al revés
POKE &HD6E7, &H23 por eso conviene trabajar en hexadecimal.

&H2328 es la dir. de inicio REAL del segundo bloque.

POKE &HD6E9, &HC8 LB → Byte bajo en dir. ram menor
POKE &HD6EA, &H32 HB→ Byte alto en dir. ram mayor.
Tiene su lógica, ¿no?

Y **&H32C8** es la longitud del mismo.

Ahora podríamos cargar el segundo bloque con **DEFUSR=&HD6E1:A=USR(0)** pero....¡¡¡ALTO!!!, ¡¡ESPERA!!!, ¡NO lo hagas todavía!.

Aunque “podamos” cargarlo (que NO podemos por estar las paginas 0 y 1 cargadas en ROM), no te parece que cargarlo en su dirección de inicio REAL (**&H2328** es ram oculta al basic) no nos sería muy útil, porque ¿a ver?, una vez cargado ¿cómo puñetas íbamos a grabarlo con BSAVE?. Tanto **BLOAD** como **BSAVE** operan con la **RAM** comprendida entre **&H8000** y **&HFFFF** porque, como es lógico, para que funcionen estas instrucciones han de estar conectados el BIOS y el BASIC (rom). Por eso, si grabas con una dirección inferior a **&H8000**, estarás grabando la rom. Y si estás pensando en cargar algo en rom...¡pégate un capón de mi parte!.

Sabemos que no podemos cargar nada por debajo de **&H8000 (32.768)** en basic; de hecho, ni siquiera podemos empezar a cargar por esa dirección puesto que precisamente en **&H8000** comienzan a almacenarse los programas Basic (y no nos olvidemos del pequeño cargador basic que deberemos hacer para cargar el Black Beard desprotegido). No recomiendo cargar nada por debajo de **&H8200**, reservando esos **512 bytes** para el cargadorcito basic.

La táctica a emplear es clara; cambiar la dirección de inicio REAL del bloque por una dirección TEMPORAL (o ficticia) que se encuentre en ram basic (desde **&H8200** hasta **&HEF00**). Por encima de **&HEF00** es peligroso cargar datos porque podrían colisionar con la **PILA BASIC**.

Para encontrar esa dirección TEMPORAL hay que tener presente la longitud de los bloques a cargar. Si la dirección escogida es muy elevada, p.ej. **&HD000**, los datos cargados colisionarían con el programa cargador y, ¡Ahora sin duda!, se nos bloquearía el ordenador. O sea, que en realidad puedes cargar los bloques del Black Beard entre las direcciones **&H8200** y **&HD6D7**. En total son **21.719 bytes libres**, un buffer de carga/grabación de un tamaño más que suficiente si consideramos que el bloque más largo del Black Beard mide **&H4FFF (20.479 bytes)**.

Vamos a asignarle al segundo bloque una nueva dirección de inicio (temporal) como p. ej. **&H8400**:

POKE &HD6E6, &H00
POKE &HD6E7, &H84

Ahora SI estamos en condiciones de cargar este bloque; cosa que haremos de inmediato como ya sabemos (DEFUSR=.....). Una vez cargado lo grabaremos, tras el primero con **BSAVE "nombre", &H8400, &H8400+&H32C8 (&HB6C8)**.

Vamos a por el tercer bloque, que al igual que el anterior tiene su inicio real en ram oculta. Habrá que cargarlo en ram basic (y, ¿por qué no a partir de **&H8400** otra vez?). ¡Hagámoslo!:

POKE &HD6E9, &H00
POKE &HD6EA, &H2A

No hace falta alterar la dirección de inicio, **&H8400**. Tranquilo si esto te suscita alguna duda –muy lógica. Más adelante volveremos sobre este tema. De momento, graba este tercer bloque con **BSAVE "nombre", &H8400, &H8400+&H2A00 (&HAE00)**.

El último bloque también se cargará a partir de **&H8400** (que es “casualmente” su dirección de inicio real). Tendremos que modificar la longitud en el programa cargador, como sabemos hacer....¡venga!, ¿a qué estás esperando para hacerlo?:

POKE &HD6E9, &HFF
POKE &HD6EA, &H4F

Cárgalo como siempre y grábalo tras los otros bloques. La desprotección, propiamente dicha ha terminado. Hemos desmontado el PUZZLE, ahora debemos montarlo y encajarlo para que todo funcione de nuevo.

Por si no tienes claro lo que hemos hecho hasta ahora, aquí te lo resalto de un modo más gráfico. Esta es la situación presente:

<u>NUESTRA GRABACIÓN</u>	<u>EL ORIGINAL</u>
1er bloque: Una pantalla de carga que no se ejecuta. &H9C40 - &HB7FA - &H9C40	Una pantalla de carga que se ejecuta, en &H9C40 - &HB7FA - &H9C40
2do. Bloque: Un bloque de datos descolocado en : &H8400 - &HB6C8	Un bloque de datos perfectamente colocado en ram oculta: &H2328 - &H55F0
3er. Bloque: Un bloque de datos alucinado en: &H8400 - &HAE00	Otro que está en su sitio en: &H55F0 - &H7FF0
4to. Bloque: El último mohicano. &H8400 - &HD3FF	La oveja negra de la familia: &H8400 - &HD3FF

Eso sin hablar del *Analizador de Slots* y del *Loader basic* que nos faltan. ¡Vale!, los resultados no son muy brillantes, lo reconozco. ¿Qué esperabas de un “Hacker” de cuarta regional como yo? (¡Ah!, los tiempos del abuelo JACK. ¡Aquellos si eran años!).

“El viejo vaquero limpió con dignidad la pernera de su gastado pantalón. Agachó la espalda, estirando su brazo hasta coger su pisoteado sombrero. Se lo puso y, levantando la vista al cielo, sintió el reseco aliento del desierto en su garganta.

“Perro” Clyde arrastraba su malherido brazo en un desesperado intento de alcanzar su revolver. El viejo vaquero bajó la cabeza. Su rostro, antaño sonrosado, su ingenio, siempre vivaz, se habían esfumado. Su semblante era la viva imagen de la muerte.

Avanzando unos pasos hacia Clyde se detuvo a su lado. Al agacharse junto a él, en cuclillas, el “perro” sintió como sus ladridos se le ahogaban en la garganta. La palidez del viejo le hizo comprender, como ninguna palabra podría haber hecho, que había llegado su hora, su minuto y su segundo.

Con un rápido gesto el viejo vaquero puso su pistola en la frente de Clyde. Su mano temblaba ligeramente, pero en el tono de su voz no se percibía ninguna duda.

- *“Dame una buena razón por la que debas vivir”- dijo.*

Y aquel disparo resonó, llevado por el eco de la justicia, en los corazones de los habitantes de Denton City.” (1)

Forastero, ¿has rezado ya tus oraciones?. Más te vale haberlo hecho, porque ahora llega....

OTRO POQUITO DE CÓDIGO MÁQUINA:

Nos volvemos a detener a medio camino (esto se va pareciendo al Talgo Almeria-Madrid) para hacer un par de precisiones:

- Si te pareció complejo, confuso o complicado lo visto hasta el momento....¡más te vale dejarlo! (y ponerte a empollar el “pequeño manual” de Rodnay Zaks y los dos capitulitos anteriores). De ahora en adelante las cosas van a complicarse cada vez más. Esto irá “in crescendo”....
- Si te has ido enterando de todo hasta el momento, quizás, solo quizás, hayas sido lo bastante perspicaz como para deducir un axioma de la programación en C/M:

“UN PROGRAMA (C/M) SOLO SE EJECUTARÁ CORRECTAMENTE SI ES CARGADO EN SUS DIRECCIONES DE MEMORIA ORIGINALES (DONDE FUE ENSAMBLADO)”.

Como casi todas las máximas, ésta es una perogrullada y, ¡para colmo!, tiene sus excepciones (el GEN-MON p. e.). Si bien es cierto existen programas que pueden cargarse –y ejecutarse- en direcciones de memoria variables, siendo conocidos por ello como “reubicables”, no es este el caso de los juegos. Por tanto deberemos, de alguna manera, devolver los bloques desplazados a sus direcciones originales (el segundo y el tercero en este caso).

Desgraciadamente solo podemos hacer tal cosa, por tratarse de desplazamientos a la ram oculta, trabajando en C/M (comprenderéis ahora el por qué de los capítulos anteriores de este impresentable cursillo). Necesitamos de un programita en ensamblador que realice esas transferencias de datos. ¡Ánimo hombre!. NO es tan difícil como parece....

Este es el programa **TRANSFER** para el **BLACK BEARD**:

```
10          ORG &HE000
15 ;TRANSFER BYTES
20 BK2:      DI
30          LD A, (&HE292)
40          OUT (&HA8), A
50          LD HL, &H8400
60          LD DE, &H2328
70          LD BC, &H32C8
80          LDIR
90          LD A, (&HE290)
100         OUT (&HA8), A
110        RET
120 BK3:      DI
130         LD A, (&HE292)
140         OUT (&HA8), A
150         LD HL, &H8400
160         LD DE, &H55F0
170         LD BC, &H2A00
180         LDIR
190         LD A, (&HE290)
200         OUT (&HA8), A
210        RET
220 BK4:      DI
230         LD A, (&HE292)
240         OUT (&HA8), A
250         JP &H92AC
```

La dirección de inicio del TRANSFER (**&HE000**) no debe colisionar con las direcciones ocupadas por el juego (**&H2328-&HD3FF**). El listado anterior debe ensamblarse con el **RSC / RSC 2**. Una vez ensamblado sin errores, hay que grabarlo en cinta. No debe ser ejecutado porque este programa **SÓLO FUNCIONARÁ SI PREVIAMENTE HAS CARGADO Y EJECUTADO EL ANALIZADOR DE SLOTS DE TOPO** (¿Te acuerdas del “**TEST**” del principio?) puesto que utiliza los datos calculados por aquel:

- **&HE292** TODO RAM.
- **&HE290** ROM / RAM.

Venga, vamos a cargar de una vez el **Black Beard desprotegido**. Lo primero es teclear y grabar el programita cargador en basic:

```
10 REM Loader BLACK BEARD DESPROTEGIDO.  
20 COLOR 15, 1, 1: SCREEN 2  
30 BLOAD"CAS:TEST",R:BLOAD"CAS:TRANSFER"  
40 BLOAD"CAS:SCREEN",R  
50 BLOAD"CAS:BLOCK2":DEFUSR=&HE000:A=USR(0)  
60 BLOAD"CAS:BLOCK3":DEFUSR=&HE(BK3):A=USR(0)  
70 BLOAD"CAS:BLOCK4":DEFUSR=&HE(BK4):A=USR(0)
```

Nota: BK3 y BK4 son las direcciones de inicio de dichas rutinillas.

Tras esto graba el **Analizador de Slots de Topo** ("TEST") y el **TRANSFER**. Después graba los bloques desprotegidos en orden. El programa ha sido desprotegido por completo y debería cargar perfectamente tanto en cinta como en disco. Ahora bien; ¿qué hace el TRANSFER?, ¿Cómo lo hace?, ¿Y cuando debe hacerlo?.

El **TRANSFER** son, en realidad, tres subrutinillas independientes en C/M (**BK2**, **BK3** y **BK4** son sus dir. de inicio y ejecución). Cada rutina debe ser ejecutada tras cargar el bloque que deba transferir (colocar en su lugar). Olvídate de momento de la subrutina BK4, muy diferente, y centrémonos en BK2 y BK3. Ambas hacen lo mismo, de igual forma, aunque en diferentes momentos y con distintos bloques de datos. Las comento.

Lo primero que hacen es desconectar las INTERRUPCIONES con:

DI

Es algo que debe hacerse **SIEMPRE** (¡qué pesaico soy!) **antes de conectar o desconectar páginas de memoria** (que es lo siguiente que hacen):

```
LD A, (&HE292)  
OUT (&HA8), A
```

Conecta la configuración de memoria **TODO RAM** (sólo si se ejecuta antes el "TEST". ¡Pero que pesaico soy!). A continuación viene lo interesante:

```
LD HL, &H8400  
LD DE, &H2328 ó &H55F0  
LD BC, &H32C8 ó &H2A00  
LDIR
```


En el registro **HL** se carga la dirección de inicio **TEMPORAL**. En el registro **DE** se carga la dirección de inicio **REAL** de los datos en ram, y en el registro **BC** se carga la **longitud** del bloque de datos a transferir.

LDIR es una instrucción de transferencia automática que hace, ella sola, todo lo que sigue:

- “Carga de bloque con incremento repetida”.
- “El contenido de la posición de memoria **HL** se carga en la posición **DE**. A continuación se incrementan **HL** y **DE** y se decrementa **BC**. Si **BC** es distinto de cero el contador del programa se decrementa en 2, y la instrucción vuelve a ejecutarse.”

O sea,

```
INI:  (DE) ← (HL)
      DE = DE + 1
      HL = HL + 1
      BC = BC - 1
      SI BC <> 0 GOTO INI
```

Nota: Estoy considerando no obstante la posibilidad de realizar transferencias a ram oculta desde el BASIC, lo que os evitaría el tener que programar en C/M el BK2 y BK3. Estas transferencias se harían usando un comando ampliado en BASIC, p.ej. **CMDLDIR (Dir Fuente, Dir Destino, Bytes Longitud)**.

(Recuerda que soñar sigue siendo gratis, razón de peso para hacerlo....). Pero hasta que los sueños se conviertan en realidades palpables, seguid usando el ensamblador tal y como os lo he explicado.....

O sea “again”, que el bloque de datos que se inicia en **HL** y mide de longitud **BC** es transferido dato a dato, a la dirección contenida en **DE**. SE dice que **LDIR** es una instrucción muy potente porque hace sola un güebo de cosas.

Las siguientes instrucciones son necesarias para restituir la configuración **ROM/RAM** y retornar al BASIC:

```
LD A, (&HE290)
OUT (&HA8), A
RET
```

Comprenderás, si entiendes como funcionan las subrutinillas Transfer, que no tenía importancia el grabar y cargar los bloques “turbo” (todos) a partir de &H8400, porque al cargarlos desprotegidos el programa Transfer se ocupará al instante de posicionarlos en sus direcciones reales de memoria.

He dejado a propósito la explicación del complejo **BK4** para el final. A estas alturas, deberías tener claro que el bloque cuarto del programa no necesita transferirse a ninguna parte. Se encuentra en su posición ram definitiva; entonces, ¿qué sentido tiene el bloque BK4?.

- 1) Antes de activar el juego BLACK BEARD es necesario activar la **configuración TODO RAM**. Para eso sirve la rutina BK4....
- 2) pero además, será necesario saltar a la dirección de ejecución del programa (JP es más o menos como GOTO). Con este salto, salto absoluto a la fuerza, perderemos definitivamente el control del sistema y este pasará al programa, como debe ser.

Se te planteará la duda; ¿cómo saber cual es la dirección de ejecución del juego?....¡oye!, ¿tú has mirado el programa cargador del Black Beard?, lo digo porque, sabiendo leerlo y con un pelín de astucia se comprende que tras cargar el ULTIMO bloque de datos lo lógico y lo “normal” es que el programa se ejecute (como en este caso hace, con **JP &H92AC**).

Hemos acabado con el NIVEL 2 (¡aleluya!). Toma buena nota, sobre todo de las subrutinas Transfer y, en general, de todo el proceso. En el futuro **TU deberás hacerlo todo solito**, incluidas las rutinillas Transfer en C/M....

TO BE CONTINUED...



4.EJERCICIOS DELCAPITULO 3 (I):

1. Consigue todos los juegos de Grenlim que puedas, en versión protegida, y lánzate a su desprotección. TODOS, ABSOLUTAMENTE TODOS, se desprotegen de la forma explicada, aún cuando no posean el Loader Standard (como p. ej. el Valkyr).

Es evidente que NO DEBES usar el copión de GrenLim. NO se trata de que nos lo den todo hecho y nos conviertan en semivegetales estúpidamente fascinados ante la pantallas, en seres no-pensantes (para eso, para comodidad,¡CÓMPRATE UN PC!). Se trata de aprender a pensar y a realizar las cosas por uno mismo. Este es el camino más duro, el más difícil; pero también es el más gratificante.

Desproteger los programas “a mano” es una labor pesada, repetitiva, ingrata....¡bien lo sé, por propia experiencia!... pero también sé que no existe alegría comparable a la que siente un programador aficionado cuando, ante una rutina o programa (por simple que sea), tiene la satisfacción de decir: “LO HE HECHO YO”. Estas cuatro palabras abren las puertas del paraíso al autentico amante de la Informática (¿Es orgullo o simple vanidad lo que encierran estas palabras?. Algo de ambos hay, no voy a negarlo, pero en mi caso prefiero creer que encierran **autoestima**, autoafirmación, simple **satisfacción** por lo conseguido. Para mi la programación es, sobre todo, un desafío que YO resumo en la frase: “¿QUÉ SERÉ CAPAZ DE HACER MAÑANA?”. Sólo aprendiendo, practicando, pensando, HOY podré recoger el guante y afrontar el mañana, y....

....¡DEMONIOS! ¿Os habéis dado cuenta de cómo me enrollo y enrollo, y me largo en un plis-plas por los cerros de Úbeda?. ¡Cosssaaa Mala.....!).

Paco Persiana.

La autoestima es la madre de la vanidad y la satisfacción la madre del orgullo. Como decía ARISTOTELES, “En el centro está la virtud”. NO creo que casi NADA tomado en su justa medida sea negativo....

P.D. En primer lugar, quiero realizar una aclaración de suma importancia... Este curso fue creado originalmente por **FRANCISCO FUENTES LORENZO**. El formato original del mismo son fotocopias escritas a máquina, y que yo **JOSÉ MANUEL SOTO**, he intentado pasar a pdf para luego aplicarle soft de OCR, pero viendo los desastrosos resultados he optado por teclearlo todo letra a letra en formato WORD, respetando fielmente el original....

Para cualquier aclaración, comentario, sugerencia, etc, etc... no dudéis en escribirme. Me gustaría que todo aquel que tenga conocimientos sobre el tema, y pueda contribuir a mejorar este curso, ampliarlo, corregirlo, etc, que se anime y me escriba. También me gustaría me escribierais para darme vuestra opinión sobre el mismo, si lo habéis leído, y tenéis intención de seguirlo, etc, etc, etc... Y aprovecho, aún a riesgo de ser pesado que por favor, todos los que tengáis material MSX por ahí que sea interesante para el resto (libros, documentación técnica, etc, etc, ...) o si sabéis de alguien que lo tenga, que lo escaneéis y lo paséis a pdf para compartirlo con el resto y que todo ese valioso material no desaparezca con el paso del tiempo!!!).

Gracias al Paco y al Robsy por animarme a llevar a cabo esta iniciativa, y a Karloch por darle difusión. Y gracias, mil veces gracias al Paco que casualidades de la vida nos hemos vuelto a “encontrar” después de mucho tiempo, mucho, mucho tiempo... Gracias y mil veces gracias Paco por tus cartas, los juegos que gracias a ti conseguí, las dudas que me resolviste, tu infinita paciencia y por haber escrito este curso en su día “exclusivamente para mi”... (ja, ja, ja... quien te iba a decir esto por aquel entonces, ¿eh?).

Va por ti, MAESTRO...!!! ☺ ☺ ☺

Antes de despedirme “definitivamente” quiero aclarar que la realización de este tercer y último capítulo ha sido muy, muy dura debido a que he tenido que conseguir los listados que incluyo como ANEXO y revisarlos y “formatearlos”.... El **LOADER ESTANDAR DE GREMLIN** lo he conseguido gracias a **Javi** (alias **Nazgul** en Hispamsx) ☺

Como veis, esto ha llegado a su “fin”... Como consejo práctico os diré que os imprimáis todo este curso, incluido su ANEXO con los listados y que lo releáis una y otra vez si es necesario... haciendo las anotaciones pertinentes y subrayando todo lo que creáis necesario... Cuando os sintáis con el valor y las fuerzas suficientes ya sabéis lo que os queda: afrontar vosotros solos el reto de lanzaros a desproteger por vuestra cuenta y riesgo...

Me gustaria que todos aquellos que os lanceis a la aventura de la desprotección y consigais desproteger algún juego lo compartais conmigo y con los demás!!!. Estaría bien que redactarais un “mini curso” explicando las peculiaridades del juego, como lo habeis abordado, etc, etc, etc... De esta forma todos podremos aprender y establecer diferentes técnicas a la hora de abordar diferentes juegos y diferentes compañías. Quizás cada compañía (Dinamic, Topo, Opera, etc, etc, etc, etc...) hayan seguido de una forma consciente o inconsciente los mismos esquemas a la hora de “proteger” sus juegos, así que os reto y os invito a que lo comprobéis, a que investigueis, a que pongáis en práctica todo esto, y a que lo COMPARTAIS.... (Si, lo sé..., sé que soy un pesado.... pero si yo no hubiera compartido todo esto con vosotros ahora sabriais un “poquito menos”, y si el resto de los usuarios no hubiera hecho lo mismo imaginad lo que hubiera pasado....). Así que os animo a que compartais todos vuestros conocimientos, juegos, programas, documentación técnica, libros, etc, etc, etc con el resto de los usuarios, a que los escaneeis y los paseis a pdf para que puedan llegar al maximo de gente posible y todo este valioso material e información no se pierda con el paso del tiempo.

jose_manuel@mixmail.com

LISTADO 1: LISTADO ANALIZADOR SLOTS DE GREMLIN RSC II MSX 1.0

1.	9000	F3	DI	
2.	9001	DBA8	IN	A, (&HA8)
3.	9003	47	LD	B, A
4.	9004	08	EX	AF, AF '
5.	9005	78	LD	A, B
6.	9006	E630	AND	&H30
7.	9008	47	LD	B, A
8.	9009	DBA8	IN	A, (&HA8)
9.	900B	E6F0	AND	&HF0
10.	900D	3D	DEC	A
11.	900E	210000	LD	HL, &H0000
12.	9011	3C	INC	A
13.	9012	D3A8	OUT	(&HA8), A
14.	9014	4F	LD	C, A
15.	9015	7E	LD	A, (HL)
16.	9016	2F	CPL	
17.	9017	77	LD	(HL), A
18.	9018	BE	CP	(HL)
19.	9019	79	LD	A, C
20.	901A	2827	JR	Z, +&H27 (&H9043)
21.	901C	1E00	LD	E, &H00
22.	901E	79	LD	A, C
23.	901F	0F	RRCA	
24.	9020	0F	RRCA	
25.	9021	E6C0	AND	&HC0
26.	9023	B0	OR	B
27.	9024	D3A8	OUT	(&HA8), A
28.	9026	3AFFFF	LD	A, (&HFFFF)
29.	9029	2F	CPL	
30.	902A	E6F0	AND	&HF0
31.	902C	B3	OR	E
32.	902D	32FFFF	LD	(&HFFFF), A
33.	9030	79	LD	A, C
34.	9031	D3A8	OUT	(&HA8), A
35.	9033	7E	LD	A, (HL)
36.	9034	2F	CPL	
37.	9035	77	LD	(HL), A
38.	9036	BE	CP	(HL)
39.	9037	79	LD	A, C
40.	9038	2809	JR	Z, +&H09 (&H9043)
41.	903A	1C	INC	E
42.	903B	7B	LD	A, E
43.	903C	FE04	CP	&H04
44.	903E	20DE	JR	NZ, -&H22 (&H901E)
45.	9040	79	LD	A, C
46.	9041	18CE	JR	-&H32 (&H9011)
47.	9043	D604	SUB	&H04
48.	9045	210040	LD	HL, &H4000
49.	9048	C604	ADD	A, &H04
50.	904A	D3A8	OUT	(&HA8), A
51.	904C	4F	LD	C, A
52.	904D	7E	LD	A, (HL)

53.	904E	2F	CPL
54.	904F	77	LD (HL), A
55.	9050	BE	CP (HL)
56.	9051	79	LD A, C
57.	9052	282B	JR Z, +&H2B (&H907F)
58.	9054	1E00	LD E, &H00
59.	9056	79	LD A, C
60.	9057	0F	RRCA
61.	9058	0F	RRCA
62.	9059	0F	RRCA
63.	905A	0F	RRCA
64.	905B	E6C0	AND &HC0
65.	905D	B0	OR B
66.	905E	D3A8	OUT (&HA8), A
67.	9060	3AFFFF	LD A, (&HFFFF)
68.	9063	2F	CPL
69.	9064	E6F3	AND &HF3
70.	9066	B3	OR E
71.	9067	32FFFF	LD (&HFFFF), A
72.	906A	79	LD A, C
73.	906B	D3A8	OUT (&HA8), A
74.	906D	7E	LD A, (HL)
75.	906E	2F	CPL
76.	906F	77	LD (HL), A
77.	9070	BE	CP (HL)
78.	9071	79	LD A, C
79.	9072	280B	JR Z, +&H0B (&H907F)
80.	9074	7B	LD A, E
81.	9075	C604	ADD A, &H04
82.	9077	5F	LD E, A
83.	9078	FE10	CP &H10
84.	907A	20DA	JR NZ, -&H26 (&H9056)
85.	907C	79	LD A, C
86.	907D	18C9	JR -&H37 (&H9048)
87.	907F	08	EX AF, AF'
88.	9080	D3A8	OUT (&HA8), A
89.	9082	08	EX AF, AF'
90.	9083	32FEFF	LD (&HFFFE), A
91.	9086	FB	EI
92.	9087	C9	RET
93.	9088	00	NOP
94.	9089	00	NOP
95.	908A	00	NOP
96.	908B	00	NOP
97.	908C	00	NOP
98.	908D	00	NOP
99.	908E	00	NOP
100.	908F	00	NOP
101.	9090	00	NOP
102.	9091	00	NOP
103.	9092	FF	RST &H38
104.	9093	80	ADD A, B
105.	9094	FF	RST &H38
106.	9095	C0	RET NZ
107.	9096	FF	RST &H38
108.	9097	E0	RET PO

109.	9098	FF	RST	&H38
110.	9099	F0	RET	P
111.	909A	FF	RST	&H38
112.	909B	F0	RET	P
113.	909C	FF	RST	&H38
114.	909D	F8	RET	M
115.	909E	FF	RST	&H38
116.	909F	FF	RST	&H38
117.	90A0	FF	RST	&H38
118.	90A1	FF	RST	&H38
119.	90A2	FF	RST	&H38
120.	90A3	FF	RST	&H38
121.	90A4	FF	RST	&H38
122.	90A5	FF	RST	&H38
123.	90A6	FF	RST	&H38
124.	90A7	F8	RET	M
125.	90A8	FF	RST	&H38
126.	90A9	F0	RET	P
127.	90AA	FF	RST	&H38
128.	90AB	F0	RET	P
129.	90AC	FF	RST	&H38
130.	90AD	E0	RET	PO
131.	90AE	FF	RST	&H38
132.	90AF	C0	RET	NZ
133.	90B0	FF	RST	&H38
134.	90B1	80	ADD	A,B
135.	90B2	CD9F8C	CALL	&H8C9F
136.	90B5	DD21848C	LD	IX,&H8C84
137.	90B9	0609	LD	B,&H09
138.	90BB	DD7E00	LD	A,(IX+&H00)
139.	90BE	B7	OR	A
140.	90BF	20F1	JR	NZ,-&H0F (&H90B2)
141.	90C1	DD23	INC	IX
142.	90C3	10F6	DJNZ	-&H0A (&H90BB)
143.	90C5	21F827	LD	HL,&H27F8
144.	90C8	010800	LD	BC,&H0008
145.	90CB	3E11	LD	A,&H11
146.	90CD	CDE68F	CALL	&H8FE6
147.	90D0	210018	LD	HL,&H1800
148.	90D3	010002	LD	BC,&H0200
149.	90D6	3EFF	LD	A,&HFF
150.	90D8	CDE68F	CALL	&H8FE6
151.	90DB	210000	LD	HL,&H0000
152.	90DE	CD748F	CALL	&H8F74
153.	90E1	0608	LD	B,&H08
154.	90E3	C5	PUSH	BC
155.	90E4	21CB91	LD	HL,&H91CB
156.	90E7	0603	LD	B,&H03
157.	90E9	C5	PUSH	BC
158.	90EA	E5	PUSH	HL
159.	90EB	110400	LD	DE,&H0004
160.	90EE	0604	LD	B,&H04
161.	90F0	E5	PUSH	HL
162.	90F1	0E08	LD	C,&H08
163.	90F3	7E	LD	A,(HL)
164.	90F4	D398	OUT	(&H98),A

165.	90F6	FF	RST	&H38
166.	90F7	0D	DEC	C
167.	90F8	C2F390	JP	NZ, &H90F3
168.	90FB	E1	POP	HL
169.	90FC	23	INC	HL
170.	90FD	10F1	DJNZ	-&H0F (&H90F0)
171.	90FF	E1	POP	HL
172.	9100	C1	POP	BC

LISTADO 2: LISTADO LOADER "ESTANDAR" GREMLIN
(Sacado del "Venom Strikes Back")
(Gracias a "Nazgul" por su ayuda ☺)

1.	D800	3E02	LD	A, &H2
2.	D802	CD5F00	CALL	&H5F
3.	D805	F3	DI	
4.	D806	DD21EDD8	LD	IX, &HD8ED
5.	D80A	110400	LD	DE, &H4
6.	D80D	3EFE	LD	A, &HFE
7.	D80F	37	SCF	
8.	D810	CD37D8	CALL	&HD837
9.	D813	30F0	JR	NC, -&H10 (&HD805)
10.	D815	DD2AEDD8	LD	IX, (&HD8ED)
11.	D819	ED5BEFD8	LD	DE, (&HD8EF)
12.	D81D	3EFF	LD	A, &HFF
13.	D81F	37	SCF	
14.	D820	CD37D8	CALL	&HD837
15.	D823	30E0	JR	NC, -&H20 (&HD805)
16.	D825	2AEDD8	LD	HL, (&HD8ED)
17.	D828	CD36D8	CALL	&HD836
18.	D82B	210040	LD	HL, &H4000
19.	D82E	2B	DEC	HL
20.	D82F	7C	LD	A, H
21.	D830	B5	OR	L
22.	D831	20FB	JR	NZ, -&H5 (&HD82E)
23.	D833	C305D8	JP	&HD805
24.	D836	E9	JP	(HL)
25.	D837	14	INC	D
26.	D838	08	EX	AF, AF'
27.	D839	15	DEC	D
28.	D83A	3E08	LD	A, &H8
29.	D83C	D3AB	OUT	(&HAB), A
30.	D83E	3E0E	LD	A, &HE
31.	D840	D3A0	OUT	(&HA0), A
32.	D842	D9	EXX	
33.	D843	019900	LD	BC, &H99
34.	D846	1687	LD	D, &H87
35.	D848	ED41	OUT	(C), B
36.	D84A	ED51	OUT	(C), D
37.	D84C	D9	EXX	
38.	D84D	DBA2	IN	A, (&HA2)
39.	D84F	1F	RRA	
40.	D850	E640	AND	&H40
41.	D852	F602	OR	&H2
42.	D854	4F	LD	C, A
43.	D855	BF	CP	A
44.	D856	C0	RET	NZ
45.	D857	CDCFD8	CALL	&HD8CF
46.	D85A	30FA	JR	NC, -&H6 (&HD856)
47.	D85C	211504	LD	HL, &H415
48.	D85F	10FE	DJNZ	-&H2 (&HD85F)
49.	D861	2B	DEC	HL
50.	D862	7C	LD	A, H
51.	D863	B5	OR	L

52.	D864	20F9	JR	NZ, -&H7 (&HD85F)
53.	D866	CDCBD8	CALL	&HD8CB
54.	D869	30EB	JR	NC, -&H15 (&HD856)
55.	D86B	069C	LD	B, &H9C
56.	D86D	CDCBD8	CALL	&HD8CB
57.	D870	30E4	JR	NC, -&H1C (&HD856)
58.	D872	3EC6	LD	A, &HC6
59.	D874	B8	CP	B
60.	D875	30E0	JR	NC, -&H20 (&HD857)
61.	D877	24	INC	H
62.	D878	20F1	JR	NZ, -&HF (&HD86B)
63.	D87A	06C9	LD	B, &HC9
64.	D87C	CDCFD8	CALL	&HD8CF
65.	D87F	30D5	JR	NC, -&H2B (&HD856)
66.	D881	78	LD	A, B
67.	D882	FED4	CP	&HD4
68.	D884	30F4	JR	NC, -&HC (&HD87A)
69.	D886	CDCFD8	CALL	&HD8CF
70.	D889	D0	RET	NC
71.	D88A	79	LD	A, C
72.	D88B	EE02	XOR	&H2
73.	D88D	4F	LD	C, A
74.	D88E	2600	LD	H, &H0
75.	D890	06B0	LD	B, &HB0
76.	D892	1818	JR	+&H18 (&HD8AC)
77.	D894	08	EX	AF, AF'
78.	D895	2005	JR	NZ, +&H5 (&HD89C)
79.	D897	DD7500	LD	(IX+&H0), L
80.	D89A	180A	JR	+&HA (&HD8A6)
81.	D89C	CB11	RL	C
82.	D89E	AD	XOR	L
83.	D89F	C0	RET	NZ
84.	D8A0	79	LD	A, C
85.	D8A1	1F	RRA	
86.	D8A2	4F	LD	C, A
87.	D8A3	13	INC	DE
88.	D8A4	1802	JR	+&H2 (&HD8A8)
89.	D8A6	DD23	INC	IX
90.	D8A8	1B	DEC	DE
91.	D8A9	08	EX	AF, AF'
92.	D8AA	06B2	LD	B, &HB2
93.	D8AC	2E01	LD	L, &H1
94.	D8AE	CDCBD8	CALL	&HD8CB
95.	D8B1	D0	RET	NC
96.	D8B2	3ECB	LD	A, &HCB
97.	D8B4	B8	CP	B
98.	D8B5	CB15	RL	L
99.	D8B7	06B0	LD	B, &HB0
100.	D8B9	D2AED8	JP	NC, &HD8AE
101.	D8BC	7C	LD	A, H
102.	D8BD	AD	XOR	L
103.	D8BE	67	LD	H, A
104.	D8BF	7A	LD	A, D
105.	D8C0	B3	OR	E
106.	D8C1	20D1	JR	NZ, -&H2F (&HD894)
107.	D8C3	3E09	LD	A, &H9

108.	D8C5	D3AB	OUT	(&HAB),A
109.	D8C7	7C	LD	A,H
110.	D8C8	FE01	CP	&H1
111.	D8CA	C9	RET	
112.	D8CB	CDCFD8	CALL	&HD8CF
113.	D8CE	D0	RET	NC
114.	D8CF	3E16	LD	A,&H16
115.	D8D1	3D	DEC	A
116.	D8D2	20FD	JR	NZ,-&H3 (&HD8D1)
117.	D8D4	A7	AND	A
118.	D8D5	04	INC	B
119.	D8D6	C8	RET	Z
120.	D8D7	3E7F	LD	A,&H7F
121.	D8D9	DBA2	IN	A,(&HA2)
122.	D8DB	1F	RRA	
123.	D8DC	A9	XOR	C
124.	D8DD	E640	AND	&H40
125.	D8DF	28F4	JR	Z,-&HC (&HD8D5)
126.	D8E1	79	LD	A,C
127.	D8E2	2F	CPL	
128.	D8E3	4F	LD	C,A
129.	D8E4	D9	EXX	
130.	D8E5	ED41	OUT	(C),B
131.	D8E7	ED51	OUT	(C),D
132.	D8E9	04	INC	B
133.	D8EA	D9	EXX	
134.	D8EB	37	SCF	
135.	D8EC	C9	RET	
136.	D8ED	00	NOP	
137.	D8EE	00	NOP	
138.	D8EF	00	NOP	
139.	D8F0	00	NOP	
140.	D8F1	6A	LD	L,D
141.	D8F2	86	ADD	A,(HL)
142.	D8F3	86	ADD	A,(HL)
143.	D8F4	00	NOP	
144.	D8F5	00	NOP	
145.	D8F6	00	NOP	
146.	D8F7	64	LD	H,H
147.	D8F8	83	ADD	A,E
148.	D8F9	84	ADD	A,H
149.	D8FA	84	ADD	A,H
150.	D8FB	84	ADD	A,H
151.	D8FC	85	ADD	A,L
152.	D8FD	87	ADD	A,A
153.	D8FE	88	ADC	A,B
154.	D8FF	6A	LD	L,D
155.	D900	FF	RST	&H38
156.	D901	00	NOP	
157.	D902	FF	RST	&H38
158.	D903	00	NOP	
159.	D904	FF	RST	&H38
160.	D905	00	NOP	
161.	D906	FF	RST	&H38
162.	D907	00	NOP	
163.	D908	FF	RST	&H38

164.	D909	00	NOP	
165.	D90A	FF	RST	&H38
166.	D90B	00	NOP	
167.	D90C	FF	RST	&H38
168.	D90D	00	NOP	
169.	D90E	FF	RST	&H38
170.	D90F	00	NOP	
171.	D910	FF	RST	&H38
172.	D911	00	NOP	
173.	D912	FF	RST	&H38
174.	D913	00	NOP	
175.	D914	FF	RST	&H38
176.	D915	00	NOP	
177.	D916	FF	RST	&H38
178.	D917	00	NOP	
179.	D918	FF	RST	&H38
180.	D919	00	NOP	
181.	D91A	FF	RST	&H38
182.	D91B	00	NOP	
183.	D91C	FF	RST	&H38
184.	D91D	00	NOP	
185.	D91E	FF	RST	&H38
186.	D91F	00	NOP	
187.	D920	FF	RST	&H38
188.	D921	00	NOP	
189.	D922	FF	RST	&H38
190.	D923	00	NOP	
191.	D924	FF	RST	&H38
192.	D925	00	NOP	
193.	D926	FF	RST	&H38
194.	D927	00	NOP	
195.	D928	FF	RST	&H38
196.	D929	00	NOP	
197.	D92A	FF	RST	&H38
198.	D92B	00	NOP	
199.	D92C	FF	RST	&H38
200.	D92D	00	NOP	
201.	D92E	FF	RST	&H38
202.	D92F	00	NOP	
203.	D930	FF	RST	&H38
204.	D931	00	NOP	
205.	D932	FF	RST	&H38
206.	D933	00	NOP	
207.	D934	FF	RST	&H38
208.	D935	00	NOP	
209.	D936	FF	RST	&H38
210.	D937	00	NOP	
211.	D938	FF	RST	&H38
212.	D939	00	NOP	
213.	D93A	FF	RST	&H38
214.	D93B	00	NOP	
215.	D93C	FF	RST	&H38
216.	D93D	00	NOP	
217.	D93E	FF	RST	&H38
218.	D93F	00	NOP	
219.	D940	FF	RST	&H38

220.	D941	00	NOP	
221.	D942	FF	RST	&H38
222.	D943	00	NOP	
223.	D944	FF	RST	&H38
224.	D945	00	NOP	
225.	D946	FF	RST	&H38
226.	D947	00	NOP	
227.	D948	FF	RST	&H38
228.	D949	00	NOP	
229.	D94A	FF	RST	&H38
230.	D94B	00	NOP	
231.	D94C	FF	RST	&H38
232.	D94D	00	NOP	
233.	D94E	FF	RST	&H38
234.	D94F	00	NOP	
235.	D950	FF	RST	&H38
236.	D951	00	NOP	
237.	D952	FF	RST	&H38
238.	D953	00	NOP	
239.	D954	FF	RST	&H38
240.	D955	00	NOP	
241.	D956	FF	RST	&H38
242.	D957	00	NOP	
243.	D958	FF	RST	&H38
244.	D959	00	NOP	
245.	D95A	FF	RST	&H38
246.	D95B	00	NOP	
247.	D95C	FF	RST	&H38
248.	D95D	00	NOP	
249.	D95E	FF	RST	&H38
250.	D95F	00	NOP	
251.	D960	FF	RST	&H38
252.	D961	00	NOP	
253.	D962	FF	RST	&H38
254.	D963	00	NOP	
255.	D964	FF	RST	&H38
256.	D965	00	NOP	
257.	D966	FF	RST	&H38
258.	D967	00	NOP	
259.	D968	FF	RST	&H38
260.	D969	00	NOP	
261.	D96A	FF	RST	&H38
262.	D96B	00	NOP	
263.	D96C	FF	RST	&H38
264.	D96D	00	NOP	
265.	D96E	FF	RST	&H38
266.	D96F	00	NOP	
267.	D970	FF	RST	&H38
268.	D971	00	NOP	
269.	D972	FF	RST	&H38
270.	D973	00	NOP	
271.	D974	FF	RST	&H38
272.	D975	00	NOP	
273.	D976	FF	RST	&H38
274.	D977	00	NOP	
275.	D978	FF	RST	&H38

276.	D979	00	NOP	
277.	D97A	FF	RST	&H38
278.	D97B	00	NOP	
279.	D97C	FF	RST	&H38
280.	D97D	00	NOP	
281.	D97E	FF	RST	&H38
282.	D97F	FF	RST	&H38
283.	D980	00	NOP	

COMENTARIOS SOBRE EL CÓDIGO DEL LOADER STANDARD DE GRENLM

Nota: Para facilitar el seguimiento del código he numerado las líneas ☺

- Las **líneas 1 y 2** activan el Screen 2 (el 2 del final de la 1ª línea es el que indica el Screen 2 ☺).
- La **línea 8** y la **línea 14** llaman a la **SubRutina General de Carga** (que se encuentra en la línea 25).
- En las **líneas 10 a 14** se **cargan los Bloques de DATOS** (tomando la longitud y la dirección de inicio de la cabecera).
- En la **línea 16** es el punto en el que hay que introducir el **RET** (para RETornar al BASIC!!!).
- Las **líneas 17 a 24** Ejecutan el Bloque Cargado:

```
LD HL, (&HD8ED)
CALL &HD836
      ....
JP HL
```

(Y hacen un enorme retardo, es decir, pierden tiempo)

```
LD HL, &H4000
BUC: DEC HL
LD A, H
OR L
JR NZ, BUC
```

(y cargan el siguiente Bloque, caso de haberlo)

```
JP &HD805
```

- Las **líneas 136 a 139** contienen los Datos de la Minicabecera.

Nota: La **línea 139** en realidad representa el Final “Real” del Loader de Grenlim.

Nota 2: Desde **&HD8F1** hasta **&HD980** los creadores del Loader Standard de Grenlim grabaron la RAM VACÍA (¡No!, no me preguntes, ¿por qué puñetas lo hicieron....?. N.p.i.). El programa NO USA direcciones de memoria o datos comprendidos entre esas direcciones de Ram citadas.

LISTADO 3: MÓDULO BASIC DE CONTROL DEL LOADER ESTÁNDAR DE GRENLIM (modificado para cargar los bloques "turbo" y grabar en BASIC).

```
10 REM COPION GRENLIM V1.0

20 KEY OFF: SCREEN 1,,0:COLOR 15, 4, 4:WIDTH 32:LOCATE
8,22:PRINT"CARGANDO C/M...":BLOAD"CAS:LOADER"

30 CLEAR 20,&H86FF:SCREEN1,,0:LOCATE 3,1:PRINT"COPION GRENLIM version
1.0":LOCATE 2,3:PRINT"_____":LOCATE 3,2:PRINT"
(c) 1991 "

40 LOCATE 7,6:PRINT"1. CARGAR BLOQUES ...":LOCATE 7,10:PRINT"2. GRABAR
BLOQUES ...":LOCATE 7,14:PRINT"3. ACABAR (end) ..."

50 LOCATE 9, 17:PRINT"ESCOGE OPCION":LOCATE 3,20:PRINT"Inicio Final
Ejecucion":LOCATE 3,21:PRINT"0000    0000    0000"

60 Z$=INKEY$:IFZ$="" THEN 60

70 IF VAL(Z$)<1ORVAL(Z$)>3 THEN 60

80 IF Z$="3" THEN CLS:KEY ON:END ELSE IF Z$="2" THEN GOSUB 110 ELSE IF
Z$="1" THEN GOSUB 90

90 REM CARGAR

100 LOCATE 7,17:PRINT"Cargando
...":DEFUSR0=&HD800:A=USR(0):LOCATE7,17:PRINTSPC(20):LOCATE9,17:PRINT"Esc
oge Opcion":GOSUB 140:RETURN 30

110 REM GRABAR

120 GOSUB 140:LOCATE 7,17:PRINTSPC(20):LOCATE
7,17:INPUT"NOMBRE:";N$:IFLEN(N$)>6 THEN 120

130 LOCATE 8,17:PRINT SPC(20):LOCATE 7,17:PRINT"GRABANDO ..." N$:BSAVE
N$, INI, FIN:LOCATE 7,

140
IN$=HEX$(PEEK(&HD8ED)+256*PEEK(&HD8EE)):LN$=HEX$(PEEK(&HD8EF)+256*PEEK(&H
D8F0)):INI=VAL("&H"+IN$):LNG=VAL("&H"+LN$):FIN=INI+LNG

150 LOCATE 4,21:PRINT IN$:LOCATE 13,21:PRINT HEX$(FIN):LOCATE
22,21:PRINTIN$:RETURN
```

Fija tu atención en las rutinas Basic de:

- **Carga:** Líneas 90-100
- **Grabación:** Líneas 110-130

Interesa también la línea 140....

LISTADO 4: LISTADO LOADER BLACK BEARD _ RSC II MSX 1.0

1.	D6D8	CD5053	CALL &HC350
2.	D6DB	CDE6D7	CALL &HD7E6
3.	D6DE	CDEBD7	CALL &HD7EB
4.	D6E1	CDF0D7	<u>CALL &HD7F0</u>
5.	D6E4	DD21409C	LD <u>IX</u> , &H9C40
6.	D6E8	11BA1B	LD <u>DE</u> , &H1BBA
7.	D6EB	3EFF	LD A, &HFF
8.	D6ED	37	SCF
9.	D6EE	CD1FD7	CALL &HD71F
10.	D6F1	CD409C	CALL &H9C40
11.	D6F4	DD212823	LD IX, &H2328
12.	D6F8	11C832	LD DE, &H32C8
13.	D6FB	3EFF	LD A, &HFF
14.	D6FD	37	SCF
15.	D6FE	CD1FD7	CALL &HD71F
16.	D701	DD21F055	LD IX, &H55F0
17.	D705	11002A	LD DE, &H2A00
18.	D708	3EFF	LD A, &HFF
19.	D70A	37	SCF
20.	D70B	CD1FD7	CALL &HD71F
21.	D70E	DD210084	LD IX, &H8400
22.	D712	11FF4F	LD DE, &H4FFF
23.	D715	3EFF	LD A, &HFF
24.	D717	37	SCF
25.	D718	CD1FD7	CALL &HD71F
26.	D71B	F3	<u>DI</u>
27.	D71C	C3AC92	<u>JP</u> &H92AC
28.	D71F	F3	DI
29.	D720	08	EX AF, AF '
30.	D721	D9	EXX
31.	D722	C5	PUSH BC
32.	D723	D5	PUSH DE
33.	D724	E5	PUSH HL
34.	D725	21A6FC	LD HL, &HFCA6
35.	D728	060C	LD B, &H0C
36.	D72A	2B	DEC HL
37.	D72B	56	LD D, (HL)
38.	D72C	2B	DEC HL
39.	D72D	5E	LD E, (HL)
40.	D72E	D5	PUSH DE
41.	D72F	10F9	DJNZ -&H07 (&HD72A)
42.	D731	7C	LD A, H
43.	D732	60	LD H, B
44.	D733	68	LD L, B
45.	D734	39	ADD HL, SP
46.	D735	31A4FC	LD SP, &HFCA4
47.	D738	1161C9	LD DE, &HC961
48.	D73B	D5	PUSH DE
49.	D73C	11D9ED	LD DE, &HEDD9
50.	D73F	D5	PUSH DE
51.	D740	11E100	LD DE, &H00E1
52.	D743	D5	PUSH DE
53.	D744	11D9CD	LD DE, &HCDD9

Texto original de: **FRANCISCO FUENTES LORENZO.**Pasado a formato WORD por: **JOSÉ MANUEL SOTO****jose_manuel@mixmail.com**

54.	D747	D5	PUSH DE
55.	D748	11ED69	LD DE, &H69ED
56.	D74B	D5	PUSH DE
57.	D74C	E5	PUSH HL
58.	D74D	D9	EXX
59.	D74E	0EA8	LD C, &HA8
60.	D750	ED60	IN H, (C)
61.	D752	A4	AND H
62.	D753	6F	LD L, A
63.	D754	CD9AFC	CALL &HFC9A
64.	D757	3841	JR C, +&H41 (&HD79A)
65.	D759	3EE4	LD A, &HE4
66.	D75B	329EFC	LD (&HFC9E), A
67.	D75E	CD9AFC	CALL &HFC9A
68.	D761	3837	JR C, +&H37 (&HD79A)
69.	D763	47	LD B, A
70.	D764	08	EX AF, AF '
71.	D765	B8	CP B
72.	D766	37	SCF
73.	D767	2031	JR NZ, +&H31 (&HD79A)
74.	D769	181C	JR +&H1C (&HD787)
75.	D76B	F1	POP AF
76.	D76C	DDE5	PUSH IX
77.	D76E	D9	EXX
78.	D76F	C1	POP BC
79.	D770	217203	LD HL, &H0372
80.	D773	09	ADD HL, BC
81.	D774	300C	JR NC, +&H0C (&HD782)
82.	D776	EB	EX DE, HL
83.	D777	21E8FF	LD HL, &HFFE8
84.	D77A	19	ADD HL, DE
85.	D77B	3805	JR C, +&H05 (&HD782)
86.	D77D	E1	POP HL
87.	D77E	E5	PUSH HL
88.	D77F	19	ADD HL, DE
89.	D780	77	LD (HL), A
90.	D781	0A	LD A, (BC)
91.	D782	02	LD (BC), A
92.	D783	D9	EXX
93.	D784	DD23	INC IX
94.	D786	1B	DEC DE
95.	D787	CD9AFC	CALL &HFC9A
96.	D78A	380E	JR C, +&H0E (&HD79A)
97.	D78C	F5	PUSH AF
98.	D78D	A8	XOR B
99.	D78E	47	LD B, A
100.	D78F	7B	LD A, E
101.	D790	D399	OUT (&H99), A
102.	D792	B2	OR D
103.	D793	3E87	LD A, &H87
104.	D795	D399	OUT (&H99), A
105.	D797	20D2	JR NZ, -&H2E (&HD76B)
106.	D799	F1	POP AF
107.	D79A	9F	SBC A, A
108.	D79B	B0	OR B
109.	D79C	08	EX AF, AF '

```
110. D79D 3EF3 LD A,&HF3
111. D79F 329EFC LD (&HFC9E),A
112. D7A2 AF XOR A
113. D7A3 CD9AFC CALL &HFC9A
114. D7A6 D9 EXX
115. D7A7 E1 POP HL
116. D7A8 F9 LD SP,HL
117. D7A9 218EFC LD HL,&HFC8E
118. D7AC 060C LD B,&H0C
119. D7AE D1 POP DE
120. D7AF 73 LD (HL),E
121. D7B0 23 INC HL
122. D7B1 72 LD (HL),D
123. D7B2 23 INC HL
124. D7B3 10F9 DJNZ -&H07 (&HD7AE)
125. D7B5 E1 POP HL
126. D7B6 D1 POP DE
127. D7B7 C1 POP BC
128. D7B8 D9 EXX
129. D7B9 08 EX AF,AF'
130. D7BA FE01 CP &H01
131. D7BC C9 RET
132. D7BD 5F LD E,A
133. D7BE E60F AND &H0F
134. D7C0 D399 OUT (&H99),A
135. D7C2 3E87 LD A,&H87
136. D7C4 D399 OUT (&H99),A
137. D7C6 37 SCF
138. D7C7 C9 RET
139. D7C8 1E13 LD E,&H13
140. D7CA 3E09 LD A,&H09
141. D7CC D3AB OUT (&HAB),A
142. D7CE 3E01 LD A,&H01
143. D7D0 D399 OUT (&H99),A
144. D7D2 3E87 LD A,&H87
145. D7D4 D399 OUT (&H99),A
146. D7D6 C9 RET
147. D7D7 2191E2 LD HL,&HE291
148. D7DA 1819 JR +&H19 (&HD7F5)
149. D7DC 2191E2 LD HL,&HE291
150. D7DF 181A JR +&H1A (&HD7FB)
151. D7E1 2191E2 LD HL,&HE291
152. D7E4 181B JR +&H1B (&HD801)
153. D7E6 2193E2 LD HL,&HE293
154. D7E9 180A JR +&H0A (&HD7F5)
155. D7EB 2193E2 LD HL,&HE293
156. D7EE 180B JR +&H0B (&HD7FB)
157. D7F0 2193E2 LD HL,&HE293
158. D7F3 180C JR +&H0C (&HD801)
159. D7F5 1603 LD D,&H03
160. D7F7 1EFC LD E,&HFC
161. D7F9 180A JR +&H0A (&HD805)
162. D7FB 160C LD D,&H0C
163. D7FD 1EF3 LD E,&HF3
164. D7FF 1804 JR +&H04 (&HD805)
165. D801 1630 LD D,&H30
```

```
166. D803 1ECF      LD    E, &HCF
167. D805 F3        DI
168. D806 7E        LD    A, (HL)
169. D807 A2        AND    D
170. D808 47        LD    B, A
171. D809 3AFFFF    LD    A, (&HFFFF)
172. D80C 2F        CPL
173. D80D A3        AND    E
174. D80E B0        OR     B
175. D80F 32FFFF    LD    (&HFFFF), A
176. D812 2B        DEC    HL
177. D813 7E        LD    A, (HL)
178. D814 A2        AND    D
179. D815 47        LD    B, A
180. D816 DBA8      IN     A, (&HA8)
181. D818 A3        AND    E
182. D819 B0        OR     B
183. D81A D3A8      OUT    (&HA8), A
184. D81C C9        RET
```

COMENTARIOS SOBRE EL CÓDIGO DEL LOADER DEL BLACK BEARD

Nota: Para facilitar el seguimiento del código he numerado las líneas ☺

- La **línea 1** ejecuta el **TEST** (Analizador de Slots).
- La **línea 4** es el punto a modificar introduciendo **NOP**, **NOP** y **DI**. La nueva ejecución será: **&HD6E1** ó **&HD6E3** (es igual)
- En la **línea 5**, **IX** contiene la dirección de inicio.
- En la **línea 6**, **DE** contiene la longitud (bytes).
- En la **línea 9**, se produce la llamada a la **Subrutina General de Carga (&HD71F)**
- Las **líneas 5, 6, 7, 8 y 9** cargan el **primer bloque** desde la cinta (presentación).
- En la **línea 10**, tras cargar la portada es necesario sacarla por pantalla (ejecutarla; cosa que se hace con esta instrucción). Dir. Ejecución = **&H9C40**.
- Las **líneas 26 y 27** ejecutan el programa:
DI
JP &H92AC
- En la **línea 28** empieza la **Subrutina General de Carga**, y ésta finaliza en la **línea 146** con la instrucción **RET**
- El resto del programa se ejecuta tras el **TEST**, caso de hacerlo, y sirve para manejar los **SLOTS**. Usa los datos almacenados en **E291** y **E293** (que presumo que serán para los **MSX2**, porque los datos para los **MSX** están como sabemos, en **E290** y **E292**).