Disk Loader

+3Dos – Entre el dolor y la satisfacción

por *Sergio Llata* (aka *thEpOpE*)

Desde la aparición del Zx Spectrum +3, los desarrolladores comenzaron a lanzar sus juegos en el formato de disco. Era el elemento diferenciador de esta máquina, ya que internamente su potencia era similar a los Zx Spectrum +2A, el equivalente del +3 pero sin unidad de disco, ni el *hardware* de la controladora de disco, ocupando su lugar un reproductor de casete integrado.

Aunque gran parte de los juegos se seguían editando en formato de cinta, la mayoría no aprovechaban al 100% las capacidades de memoria, haciendo que los juegos se cargasen por fases.

Solamente unas pocas compañías editaron discos que contenían los nuevos juegos, y alguna selección de sus juegos míticos. Muchos de estos, además, eran juegos que originalmente fueron diseñados para los modelos de 48k.

El origen

Toda esta tarea comenzó como una consulta que me hizo el equipo Pat Morita[1], para hacer la versión `Dsk` (imagen de disco) de su juego TokiMal para Zx Spectrum +3. Había hecho ya un cargador que aprovechaba la compresión con la rutina zx0[2], y por tanto ya conocía algunas de las dificultades que planteaba este juego en cuestión.

El primer escollo es que el Mk2[3] sitúa el inicio del código en una zona baja de memoria. El código empieza en la dirección `24100`, y por tanto tenemos muy poco espacio para poder crear el cargador Basic. Hay que tener en cuenta que si situamos la pila en `24099` y el Basic empieza en `23755`, apenas nos quedan 300 bytes para hacer el cargador. Aunque pueda parecer mucho espacio, pensemos que una linea como "`10 BORDER 0: PAPER 0: INK 0: CLEAR 24099`" ya ocupa 44 bytes una vez es codificada en memoria.

Otra de las dificultades que se plantean es que el juego carga datos en todas las páginas de Ram del modelo de 128k. El bloque principal del juego son `37331` bytes que se sitúan a partir de la dirección `24100`, y que por tanto ocupan las páginas 5, 2, y 0 (las que se mapean en el momento de ejecutar el Basic[4]), quedando libre la parte más alta de la memoria (desde `61431` hasta `65535`). Esta zona posteriormente es usada por el *engine* del juego para situar tablas y datos durante la ejecución del mismo.

El espacio que se ocupa con todos los bloques del juego, es el siguiente:

```
Ram 0 : 49152 - 12279 bytes
Ram 1 : 49152 - 10693 bytes
Ram 2 : 32768 - 16384 bytes
Ram 3 : 49152 - 15464 bytes
Ram 4 : 49152 - 15997 bytes
Ram 5 : 24100 - 8668 bytes
Ram 6 : 49152 - 12999 bytes
Ram 7 : 49152 - 5030 bytes
```

Hay que tener en cuenta que la `Ram 5` comienza en `16384`, y es la que tiene la memoria de video, las variables de sistema, el

1 https://greenwebsevilla.itch.io/

2 Rutina desarrollada por Einar Saukas. https://github.com/einar-saukas/ZX0

3 https://github.com/mojontwins/mk2

4 En el momento de ejecutar el Basic, los modelos +3 sitúan la `Rom 4` en la zona de memoria `0` a `16383`; la `Ram 5` se sitúa ocupando de `16384` a `32767`, la `Ram 2` en el rango de `32768` a `49151` y la `Ram 0` en el rango superior, de `49152` a `65535`.

Basic, la pila, y los primeros 8kb del código del juego.

Además, en el proceso de carga hay que añadir los 6912 bytes de la pantalla de carga, que en principio se cargan en 16384 (Ram 5).

Grandes diferencias de los cargadores de cinta y de disco

Un cargador de cinta para este caso no exige un esfuerzo especial, más allá de hacer el proceso de paginar y llamar a la rutina que carga de cinta para que cargue cada bloque en su lugar. Cuando se tiene la Ram 0 paginada, se puede cargar de una vez el bloque que rellena las página 5 y 2 de Ram (podemos entender que este sería el modo de memoria de 48k).

El cargador de cinta se podría hacer incluso en Basic, pero en este caso no nos va a ser posible por el poco espacio libre que tenemos para el Basic.

Tanto en la versión Tap de descarga digital como en la versión física del juego se ha tenido que optar por cargadores hechos desde código máquina, con la técnica de incluirlos como datos codificados en la primera línea del Basic.

Estos cargadores esencialmente hacen lo siguiente:

```
Cargar 6912 bytes en 16384
Paginar Ram 1
Cargar 10693 bytes en 49152
Paginar Ram 3
Cargar 15464 bytes en 49152
Paginar Ram 4
Cargar 15997 bytes en 49152
Paginar Ram 6
Cargar 12999 bytes en 49152
Paginar Ram 7
Cargar 5030 bytes en 49152
Paginar Ram 0
Cargar 37331 bytes en 24100
```

En el caso del cargador de la edición física, se ha comprimido el código en la medida de lo posible para reducir lo más posible el tiempo de carga.

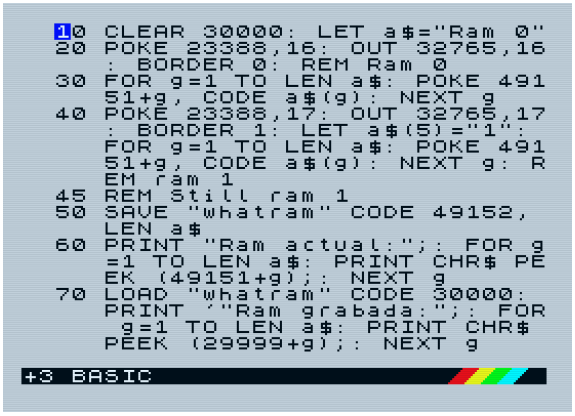
La ventaja de tener comprimidos los bloques es que se cargan menos bytes en una zona genérica de la Ram, y después se pagina la Ram correspondiente para descomprimir en ella y dejar cada bloque en su página de Ram correspondiente.

En el cargador de disco no podemos hacer esto mismo, ya que el sistema de carga de disco no es directo a la memoria, como ocurre en la cinta.

Podríamos hacer un cargador Basic, que cargue archivos de disco sin dificultad, pero estos archivos no se van a cargar en la Ram que tengamos en ese momento paginada, ya que las rutinas de grabación de carga, aunque sea sobre la unidad de disco, si se invocan desde Basic lo hacen siempre con la página de Ram 0.

Un modo sencillo de verificarlo es con el siguiente programa escrito en Basic.

```
10 CLEAR 30000: LET a$="Ram 0"
20 POKE 23388,16: OUT 32765,16
   : BORDER 0: REM Ram 0
30 FOR g=1 TO LEN a$: POKE 491
   51+g, CODE a$(g): NEXT g
40 POKE 23388,17: OUT 32765,17
   : BORDER 1: LET a$(5)="1":
   FOR g=1 TO LEN a$: POKE 491
   51+g, CODE a$(g): NEXT g: R
   EM ram 1
45 REM Still ram 1
50 SAVE "whatram" CODE 49152,
   LEN a$
60 PRINT "Ram actual:";: FOR g
   =1 TO LEN a$: PRINT CHR$ PE
   EK (49151+g);: NEXT g
70 LOAD "whatram" CODE 30000:
   PRINT '"Ram grabada:";: FOR
    g=1 TO LEN a$: PRINT CHR$
   PEEK (29999+g);: NEXT g
```



Primero pagina Ram 0, y escribe la cadena "Ram 0" en esa página de memoria. Después cambia la cadena para que contenga "Ram 1", se pagina la Ram 1, y escribe la cadena en la página de memoria.

A continuación, teniendo aún la página Ram 1 activa, graba en disco el contenido de memoria donde se ha escrito la cadena.

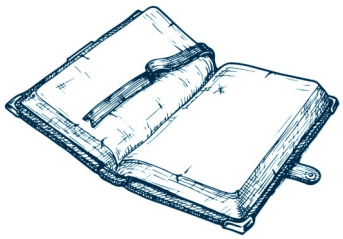
Tras terminar de grabarlo en disco se lee la cadena de memoria para ver si aún permanece la Ram 1, y finalmente hace lo

mismo con el archivo que se grabó en disco, cargándolo y mostrándolo en una zona de memoria que no se ha paginado.

Esto nos permite ver que cuando se llama a la rutina de grabar en disco desde Basic, éste graba de la `Ram 0`, y a la vuelta del +3Dos, deja la página `Ram 0` activa.

La conclusión es clara: si necesitamos cargar en páginas de Ram distintas de la 0 no podemos usar un cargador hecho únicamente en Basic.

Códices aún por descubrir



Aunque el manual del Zx Spectrum +3 es un manual con gran cantidad de información técnica, en lo referente a la descripción del +3Dos, usado a bajo nivel, tiene algunas lagunas.

A nivel genérico las rutinas del +3Dos nos permiten abrir y cerrar archivos (hasta un máximo de 16), leer su contenido, cambiar la posición de lectura y escritura en el archivo, renombrarlos, borrarlos, etc.

El puntero que indica la posición de lectura en un archivo es de 24 bits, por lo que se podría referenciar hasta 16Mb.

Algunos detalles importantes que no quedan especialmente clarificados en el manual:

1) Aunque se tenga la unidad A, o incluso la unidad B, siempre está activa la unidad M (Disco Ram), y por tanto tiene una zona de datos reservada para ella. Hay unas rutinas en el +3Dos que nos permiten reducir el espacio reservado para el disco M.

2) Los nombres de archivo se terminan con el byte `255` (`$FF`). No es necesario rellenar el nombre con espacios (*padding*), por lo que podemos usar nombres como "1.bin" sin dificultad, pero siempre terminando la cadena con el byte `255`.

3) El +3Dos lee sectores de disco mediante unas zonas de memoria intermedia (*caché*), de las que posteriormente va copiando a la zona donde han de ubicarse los datos que ha leído del disco. Cuando se acaba de reiniciar el equipo son 8 bloques de 512 bytes (4 kb), y todos ellos al inicio de la `Ram 1`.

4) Para llamar a las rutinas de +3Dos es necesario tener la `Ram 7` paginada, ya que en ella están las estructuras de archivos abiertos, posiciones de lectura o escritura en cada archivo, usos de la *caché*, etc. Cualquier modificación de determinadas zonas de la `Ram 7` puede dañar la información de qué archivos están abiertos, etc.

5) Tanto las funciones de leer como escribir datos en un archivo permiten indicar desde qué dirección de memoria se debe hacer, y qué página de Ram queremos tener activa. Son estas rutinas las que se encargan de transferir los datos de la *caché* a las direcciones y páginas correctas, para lo que usan una zona de memoria muy específica de 32 bytes.

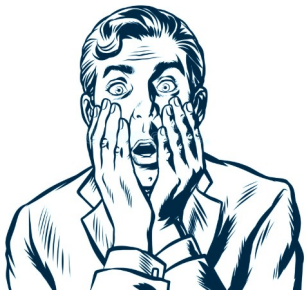
El punto 1 y 5 están muy relacionados, ya que según el manual:

Las páginas 1, 3, 4 y 6 de la RAM son tratadas como una sucesión de 128 tampones sectoriales (numerados del 0 al 127), de 512 bytes cada uno. El disco de RAM y el caché ocupan dos zonas (contiguas) de esta sucesión; su tamaño y posición son definidos en la inicialización, pero es posible redefinirlos más tarde. Los tampones no ocupados por el disco de RAM y el caché están disponibles para cualquier otro propósito. Al modificar el tamaño o la posición del disco de RAM se borran todos sus ficheros[5].

5 Manual en castellano del Zx Spectrum +3, p. 238.

Según esta descripción del manual, la *caché* de lectura de disco y el Disco Ram están ocupando por completo las cuatro páginas de memoria Ram.

Teniendo en cuenta que la Ram 7 es donde el +3Dos alberga las estructuras de datos necesarios para manejar las unidades y los archivos abiertos, ¿qué nos queda entonces? Pues apenas nada: las páginas 5, 2 y 0. Es decir, la misma configuración que cuando el equipo está en modo 48kb.



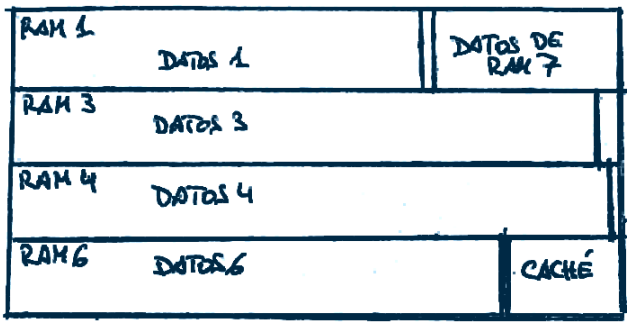
Por suerte, el manual nos ha dejado claro que es posible redefinir esos valores. Pero antes debemos saber primero dónde se ubica cada cosa, para poder eliminar el disco Ram, y posteriormente poder dejar el tamaño y el lugar de la *caché* en un lugar donde nos permita operar, y no interfiera con los datos que tenemos que cargar.

En concreto, en el esquema del TokiMal, las dos páginas de Ram que más espacio disponible tienen son la página 1 y la página 6. Cualquiera de ellas nos puede servir para dejar al final de las mismas la caché que necesita el +3Dos.

No obstante hay aún un problema grave por salvar, ya que necesitamos cargar 5030 bytes en la página Ram 7, y en esta página 7 no podemos meter datos mientras necesitemos acceder a las rutinas del +3Dos, ya que podríamos pisar y destruir estructuras de datos. Por ello será necesario cargar esos 5030 bytes en otra página de Ram que tenga esa capacidad disponible. Ya después, una vez que no tengamos que hacer uso de las rutinas de +3Dos, podremos hacer la transferencia de los datos a la Ram 7 con la seguridad de que todo funcione bien, ya que los datos que quizás estemos destruyendo del +3Dos, no se van a necesitar con posterioridad.

En la estructura del juego los datos que van en la Ram 1 (10693) más los 5030 bytes de los datos que van en la Ram 7, son un total de 15723 bytes, lo cual aún nos deja un margen de más de 600 bytes disponibles en esa página.

En la página de Ram 6, que es la última que usa para la *caché* y el Disco Ram, en el TokiMal hay que cargar 12999 bytes, lo que deja aún un margen de 3385 bytes. Ya que cada "tampón sectorial" de la *caché* es de 512 bytes, podríamos reservarnos 6, teniendo así un total de 3kb (3072 bytes).



Según las especificaciones del manual, estos "tampones" van numerados del 0 al 127, y son contiguos, por lo que del 0 al 31 son los que están en la Ram 1, del 32 al 63 los que están en la Ram 3, del 64 al 95 los que están en la Ram 4, y del 96 al 127 los que están en la Ram 6. Ya que queremos la parte más alta de la Ram 6, necesitaremos reservar los 6 últimos. Es decir, del 122 al 127 (ambos incluidos).

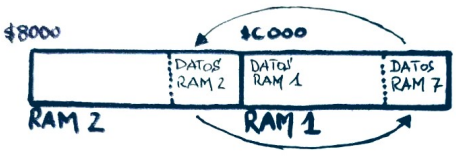
```
Distribución lineal de los "tampones"
  Ram 1 : 0 al 31
  Ram 3 : 32 al 63
  Ram 4 : 64 al 95
  Ram 7 : 96 al 127
```

Aún nos queda una pequeña dificultad, ya que tenemos los datos de la Ram 7 cargados en la Ram 1, y para poder copiarlos no podemos tener paginadas simultáneamente ambas páginas de Ram, ya que el modo de paginación hace que la página seleccionada se sitúe siempre en el rango de memoria de 49152 a 65535 ($C000 - $FFFF).

Hay varias soluciones posibles. Una de ellas sería utilizar los modos de paginación especiales que tienen los modelos +2A y +3 del Zx Spectrum, y que permiten tener 4 páginas de Ram paginadas simultáneamente, ocupando así todo el rango de memoria accesible por el z80. Sin embargo esto exige que el programa que está en ejecución asegure determinadas condiciones para que siempre esté accesible por el z80, ya que de otro modo provocaría un cuelgue.

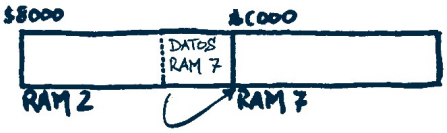
> *“El modo de paginación hace que la página seleccionada se sitúe siempre en el rango de 49152 a 65535*

El modo que elegí fue utilizar la página de Ram 2, que está situada en el rango 32768 a 49151 ($8000-$BFFF) como memoria de intercambio. Es decir, se pagina la Ram 1, y se intercambian los datos de la pagina 1 con los de la página 2 (solo los que hay que pasar a la Ram 7).

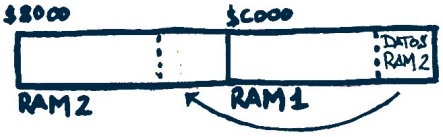


De este modo un grupo de datos de la Ram 2 ha quedado ahora en la Ram 1, y los de la Ram 1 que deben ir a la Ram 7 han quedado en la Ram 2.

Después se pagina la Ram 7, y se copian esos datos desde Ram 2 en ella.



Finalmente se pagina de nuevo la Ram 1, y se recuperan los datos que pertenecen a Ram 2, copiándolos de nuevo en el lugar que les corresponde.



Una vez que se han colocado todos los datos en su lugar ya solo queda ejecutar el programa y disfrutar del juego, que en este caso comienza en 24100.

Los problemas crecen y desaparecen

Todo parecería que está ya terminado, pero aún quedaba una sorpresa que nos produce el +3Dos: el tamaño de la pila.

Durante la ejecución de las funciones de +3Dos la pila ha de tener al menos una capacidad de 100 bytes, por lo que el tamaño que habíamos calculado de inicio para poder tener el cargador, se ve reducida aún más.

El resultado es que al crecer la pila puede pisar las partes finales del cargador. Ocurre especialmente en las funciones que van a leer datos. Para evitar este problema solo hay dos opciones: optimizar y reducir el código al máximo posible, o dejar al final del cargador código de inicio que solo se ejecute una vez, y que una vez ejecutado se pueda destruir.

Para reducir el código en parte se optó por tener todos los datos binarios concatenados en un solo archivo, para hacer solamente una apertura de archivo, y así ir leyendo secuencialmente los bloques de datos necesarios. El archivo con todos los datos concatenados son en total `104426` bytes.

El cargador Basic apenas consta de una linea `Rem` con todo el código máquina codificado en ella, y una segunda línea que ejecuta un `Randomize Usr` a la rutina cargadora. Para evitar problemas de espacio con la pila, se dejó el posicionamiento de la pila y el borrado de pantalla en el código máquina.

Para que el cargador sea ejecutado desde la opción "Cargador" del menú del Zx Spectrum +3 es necesario que el cargador tenga cabecera Basic, con autoejecución, y grabado en el disco con el nombre "Disk".

En mi caso para generar todo el cargador opté por usar el ensamblador sjasmplus[6], y generar un archivo `bin`.

Pero los archivos de disco para que sean identificados como Basic han de tener una cabecera, que indica si son archivos de código, de Basic, o de matrices. Mediante las herramientas Taptools[7] es posible añadir cabecera al archivo mediante el comando `specform`, y crear el archivo final `Dsk`, mediante el comando `mkp3fs` con los archivos que se indiquen.

Quedaba aún un paso intermedio, ya que `specform` solamente crea la cabecera de tipo `Code`, y teníamos que indicar que es un Basic con autoejecución. Por suerte el manual tiene toda la información necesaria, de qué bytes del archivo había que cambiar, y cómo calcular el byte de *checksum* con el que termina la cabecera[8].

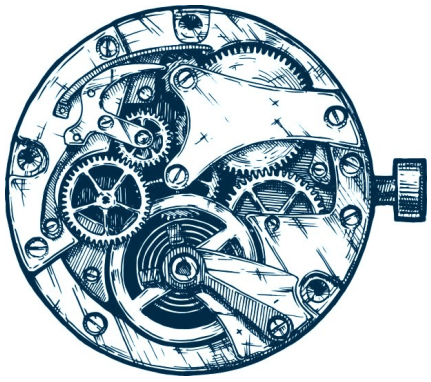
En la siguiente tabla están indicados los valores de la cabecera +3Dos dentro de la propia cabecera del archivo.

```
504C555333444F531A0100B3010000 00
33010000330100 00000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000 94
```

El primer `$00` de la cabecera indica que es un programa Basic. El siguiente valor (`$0133`)[9] es el tamaño del programa Basic, el siguiente valor (`$0000`) el número de línea de autoejecución[10], y el siguiente valor es el tamaño del Basic descontando las variables. En este caso el Basic no incluye ninguna variable con valor y por ello es el mismo valor que el tamaño del Basic.

El último byte que aparece en la cabecera (`$94`) ha de coincidir con la suma de 8 bits de los 127 bytes previos, ya que de otro modo todos estos datos no se identificarían como una cabecera, sino como el inicio de un archivo binario sin cabecera.

La creación de esta cabecera también es posible hacerlo desde el código fuente ensamblador para que lo genere el propio sjasmplus.



6 https://github.com/z00m128/sjasmplus

7 https://github.com/windenntw/taptools

8 En las páginas 226 y 227 del manual en español se describe el formato de la cabecera. La cabecera +3Dos y su significado está descrita en la página 241.

9 Los valores de 16 bits en el z80 se guardan insertando primero el byte menos significativo y después el más significativo, por eso en memoria aparece `$33` y `$01`.

10 El valor $8000 indicaría que no tiene autoejecución.

¡ Extra, Extra !
¡Aparece el código ensamblador!

```
       DEVICE ZXSPECTRUM48

DOS_ABRIR      EQU     $0106
DOS_LEER       EQU     $0112
DOS_SET_1346   EQU     $013f
DOS_OFF_MOTOR  EQU     $019c

       org 23755
linea0:
       db 0,0          ;numero de linea
       dw size_line0 - 4      ;tamaño
       db 234          ;Token de REM

start_loader:
       ld sp, 24099
       call init_state        ;código destruible
       ld bc, $0001    ;B: fichero 0
                       ;C: modo de lectura
       ld de, $0001    ;ha de existir el archivo
       ld hl, nombre
       call DOS_ABRIR
       ld bc, $0000    ;B: fichero
                       ;C: ram a paginar
       ld de, 6912     ;bytes a leer
       ld hl, 16384    ;direccion donde cargar
       call DOS_LEER
       ld bc, $0001
       ld de, 10693    ;ram1
       ld hl, 49152
       call DOS_LEER
       ld bc, $0003    ;ram3
       ld de, 15464
       ld hl, 49152
       call DOS_LEER
       ld bc, $0004    ;ram4
       ld de, 15997
       ld hl, 49152
       call DOS_LEER
       ld bc, $0006    ;ram6
       ld de, 12999
       ld hl, 49152
       call DOS_LEER
       ld bc, 0001     ;ram1
       ld de, 5030     ;5030 bytes para ram7
       ld hl, 60506    ;65536-5030
       call DOS_LEER
       ld bc, 0000     ;ram0
       ld de, 37331
       ld hl, 24100
       call DOS_LEER
       call DOS_OFF_MOTOR
       call mainrom    ;rom interprete 48k
       ld c, 1
       call setram     ;paginamos ram1
       ld hl, 60506    ;65536 - 5030
       ld de, 32768    ;esto es de la ram2
       ld bc, 5030
       call swapmem
       ld c, 7         ;paginamos ram7
       call setram
       ld hl, 32768    ;copiamos de ram2
       ld de, 49152    ;a ram7
       ld bc, 5030
       ldir            ;copia directa, sin swap
       ld c, 1
       call setram     ;paginamos ram1
       ld hl, 60506
       ld de, 32768    ;para reintegrar a ram2
       ld bc, 5030
       call swapmem    ;haciendo swap de nuevo
       ld c, 0
       call setram     ;paginamos ram0
       jp 24100        ;a jugaarrr

nombre:
       db "bin.bin",$ff

;intercambia dos bloques de memoria
swapmem:
       ld a,(hl)
       ex af,af'
       ld a,(de)
       ld (hl),a
       ex af,af'
       ld (de),a
       inc hl
       inc de
       dec bc
       ld a,b
       or c
       jr nz, swapmem
       ret

;en C se indica la pagina de ram
setram:
       ld a,(23388)
       and %11111000
       or c
       ld bc,$7ffd
       out (c),a
       ld (23388),a
       ret

mainrom:
       ld a,(23388)
       set 4,a
       ld bc,$7ffd
       out (c),a
       ld (23388),a
       ld a,(23399)
       res 0,a
       set 2,a
       ld b,$1f        ;bc=$1ffd
       out (c),a
       ld (23399),a
       ret

;el siguiente codigo solo se ejecuta al inicio
;y puede ser pisado por la pila posteriormente
init_state:
       xor a
       out (254),a     ;borde negro
       ld hl,$5800     ;zona de atributos
       ld (hl),a
       ld de,$5801
```

```
        ld bc,767
        ldir            ;atributos en negro
        di
        ld a,(23388)
        ld bc, $7ffd
        and %11101000   ;seleccion de Rom0
        or  %00000111   ;seleccion de Ram7
        out (c),a
        ld (23388),a
        ld b,$1f        ;bc=1ffd
        ld a,(23399)
        and %11111000
        or  %00000100   ;seleccion de Rom2
        out (c),a
        ld (23399),a
        ld hl,$7A00     ;H 122, L 0
                        ;6 ultimos "tampones"
                        ;de la pagina ram6
                        ;0 "tampones" de Disco Ram
        ld de,$7A06     ;H 122, L 6
                        ;6 "tampones" de cache
                        ;3kb
        jp DOS_SET_1346

size_line0 = $ - linea0

linea10:
        db 0,10
        dw size_line10 - 4
        db 242          ;PAUSE
        db 192          ;USR
        db '.',14,0,0   ;shortnumber
        dw start_loader
        db 0            ;fin del shortnumber
        db 13
size_line10 = $ - linea10

sizeofbasic = $ - 23755

        SAVEBIN "disk", 23755, sizeofbasic
```

crAck & prAy!
Sergio Llata (aka ***thEpOpE***)

Twitter: @sergio_thepope
Telegram: @thEpOpE

Agradecimientos al grupo de Telegram:
@EnsambladorZXSpectrum