

TUTORIAL DE INICIACION A LA PROGRAMACION EN LENGUAJE ENSAMBLADOR PARA MSX

1ª PARTE - EL ENTORNO DE DESAROLLO

Hola a tod@s me he decidido a escribir este tutorial para todos aquellos que queréis empezar con esto de la programación para MSX, y no sabéis por donde coger el hilo o que herramientas utilizar.

Si Bien no se va explicar paso a paso el lenguaje ensamblador del Z80 si se irán explicado nociones sobre este lenguaje encaminadas a la realización de videojuegos para nuestros MSX's.

Lo primero que os voy a proponer son las herramientas que yo me he montado para poder realizar esta labor de la manera mas fácil, lógicamente vosotros podréis sentirlos mas a gusto con otras o simplemente cambiarlas por alguna que ya estéis utilizando.

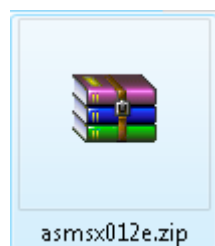
Baja este [Pack](http://www.megaupload.com/?d=9VS82B9A) de recopilación de mis herramientas desde este enlace.
<http://www.megaupload.com/?d=9VS82B9A>

Crea una carpeta en tu disco duro y llámala C:\MSX y descomprime el [PACK](#) de mis herramientas dentro de esa carpeta para tenerlas localizadas todas a mano.

Para compilar el código en ensamblador que crearemos vamos a utilizar, el que para mi es el mejor compilador cruzado de Z80 para MSX que no es otro que el [asMSX](#) de KAROSHI.

<http://karoshi.auic.es/index.php?topic=834.0> (Podéis bajar desde aquí la última versión)

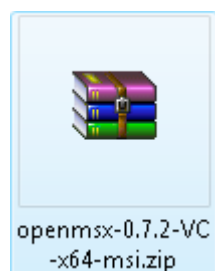
Este compilador no incluye un editor de texto que es donde escribimos el código que queremos compilar, así que buscando y buscando he decidido quedarme con el [EditPlus](#) versión 3.01 que os incluyo en el [Pack](#) y que será el punto de partida para todo nuestro desarrollo.



Vamos al directorio [MSX](#) donde tenemos el [Pack](#) y descomprime el fichero asmsx012e.zip y descomprímelo dentro de la carpeta [C:\MSX](#) dentro de esta carpeta te creara el descompresor la carpeta [asmsx012e](#)

[C:\MSX\asmsx012e](#)

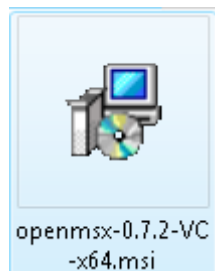
Aquí en este directorio estará nuestro Ensamblador con todos sus ejemplos.



Ahora vamos a descomprimir el Emulador de nuestro querido MSX donde probaremos nuestro código una vez compilado, lógicamente para mí el mejor emulador es el [BlueMSX](#) pero para lanzar y probar el compilado sobre el emulador uso el [OpenMSX](#) por la facilidad de lanzar o abrir el fichero compilado con el emulador.

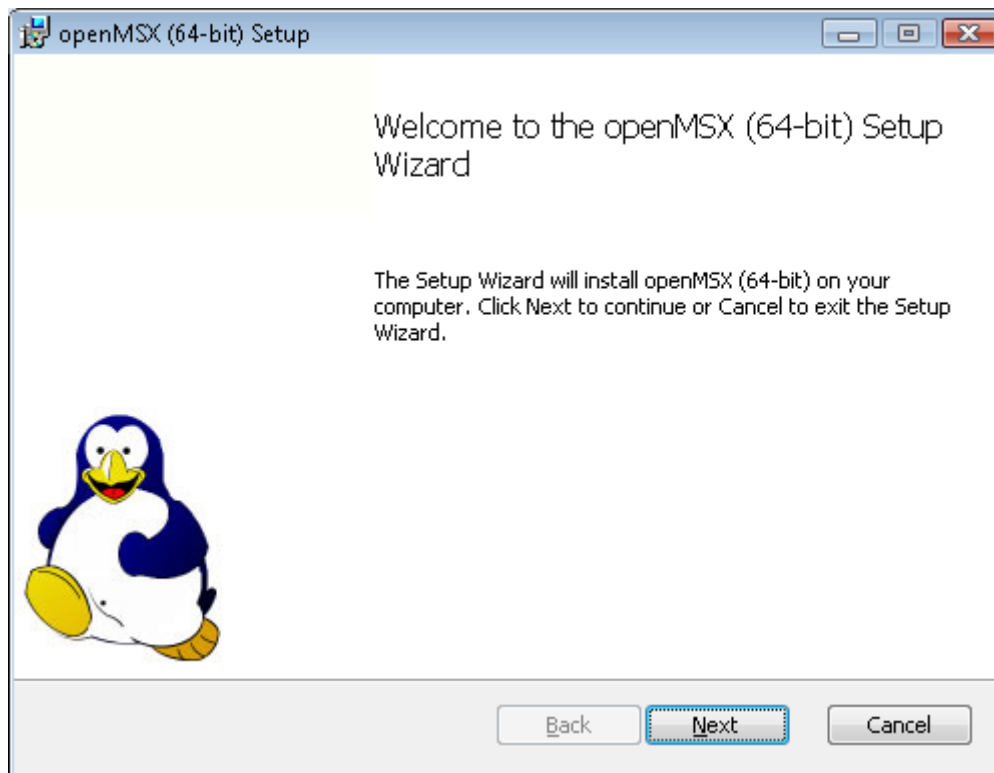
Pero para probarlo bien y depurar el código en busca de errores uso el [BlueMSX](#).

Vamos a descomprimir el OpenMSX dentro de la carpeta del Pack, veras que hay dos ficheros uno [-x64](#) si usas Windows de [64Bits](#) y [-x86](#) si usas Windows de [32 Bits](#).

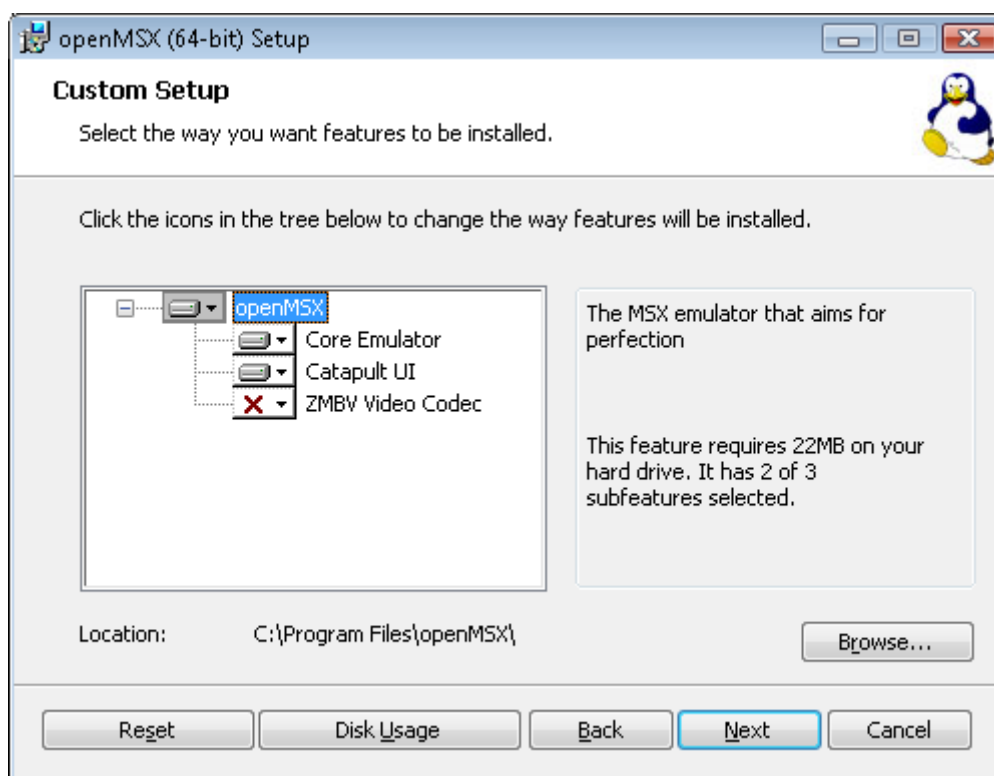


Después de que hayas descomprimido el OpenMSX según tu versión de Windows veras dentro de la carpeta del Pack, el instalador del openMSX versión 0.7.2 en mi caso uso Windows 7 Ultimate 64 Bits por eso he descomprimido la versión de 64 Bits.

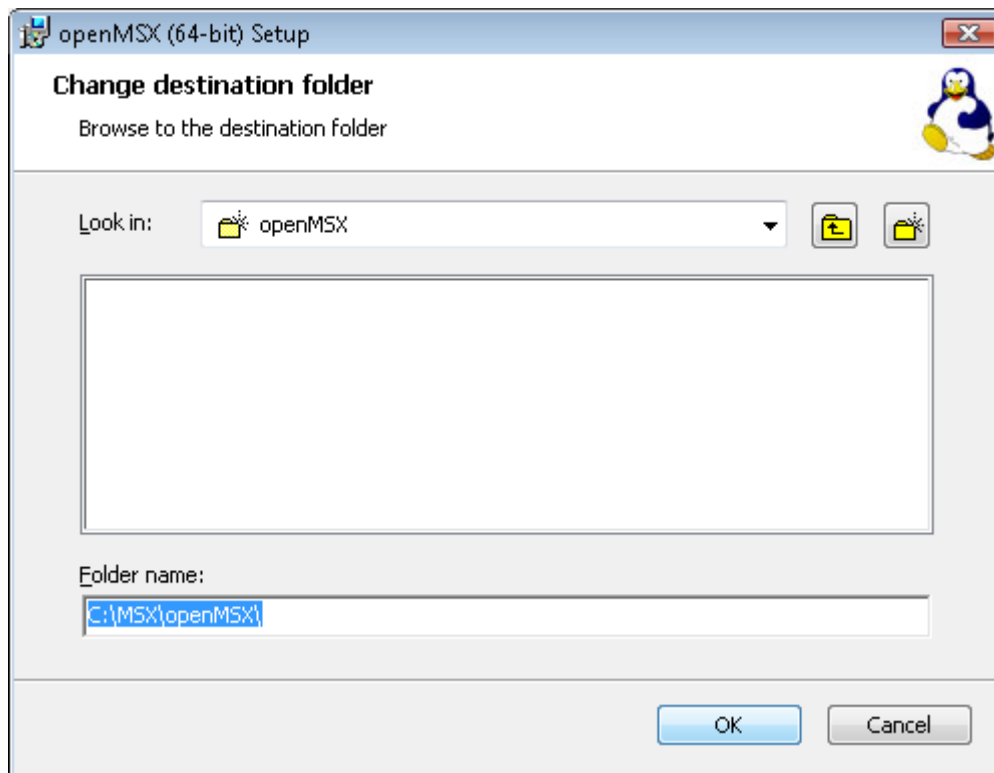
Pulsa doble click sobre el icono para que comience la instalación.



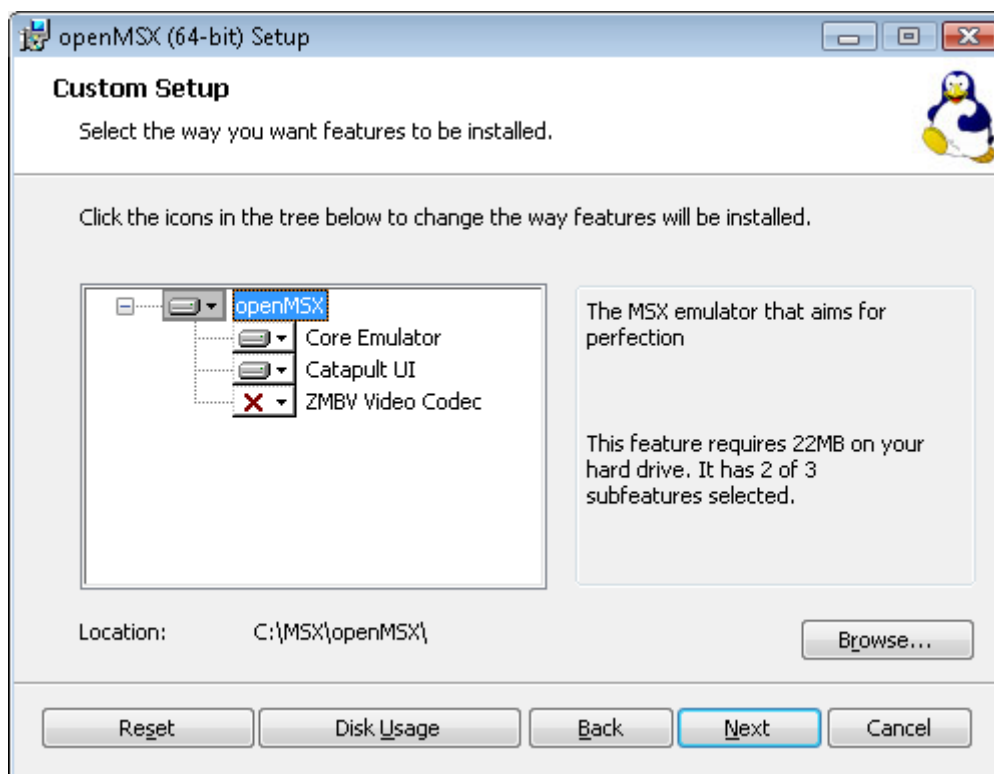
Este el instalador del [OpenMSX](#) pulsamos en el botón “Next”



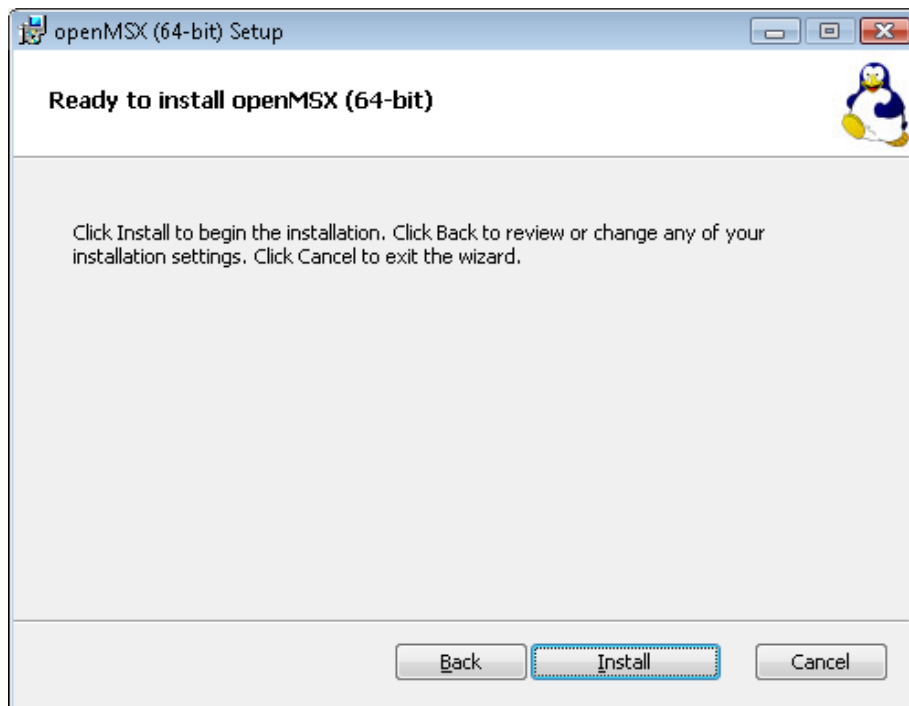
Esta parte es Importante cambiar el directorio de Instalación para que lo instale dentro del directorio [c:MSX](#) ya que después en las macros de nuestro editor lo buscaremos ahí dentro. Pulsa en “Browse”



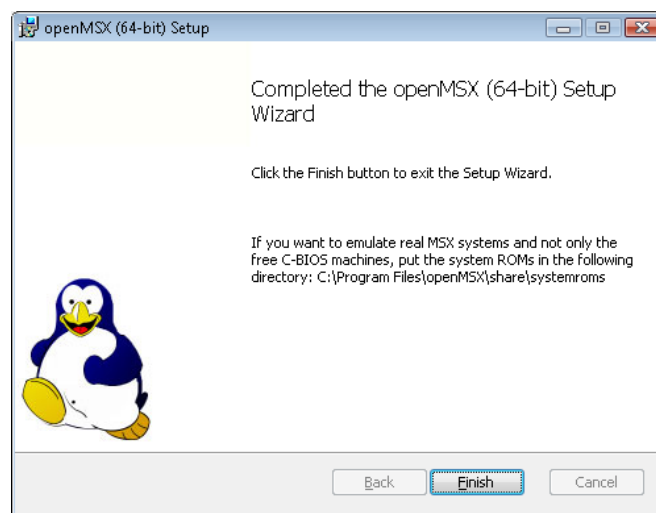
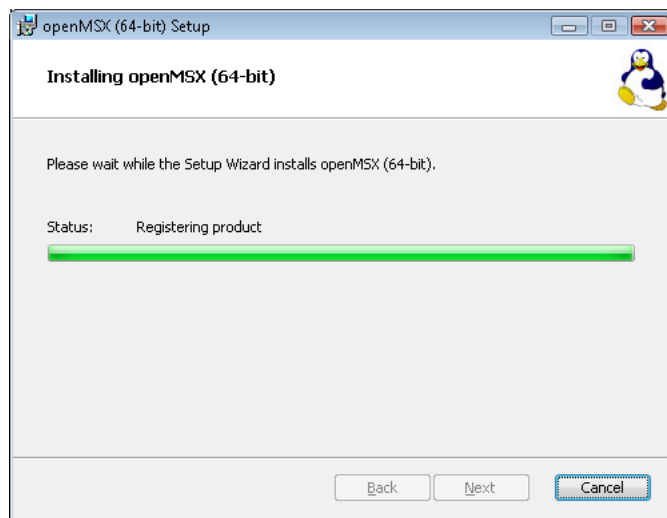
Aquí modifica el **Folder name**: y escribe **C:\MSX\openMSX** y pulsa el botón “OK”



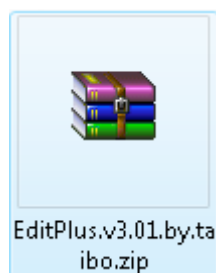
Aquí puedes ver que el directorio ha cambiado y pulsamos el botón “Next”



Finalmente pulsamos en el botón “**Install**”



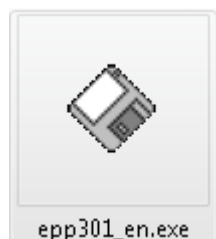
Esperas a que se instale, veras estas imágenes cuando pulses el botón “**Finish**” ya estará instalado.



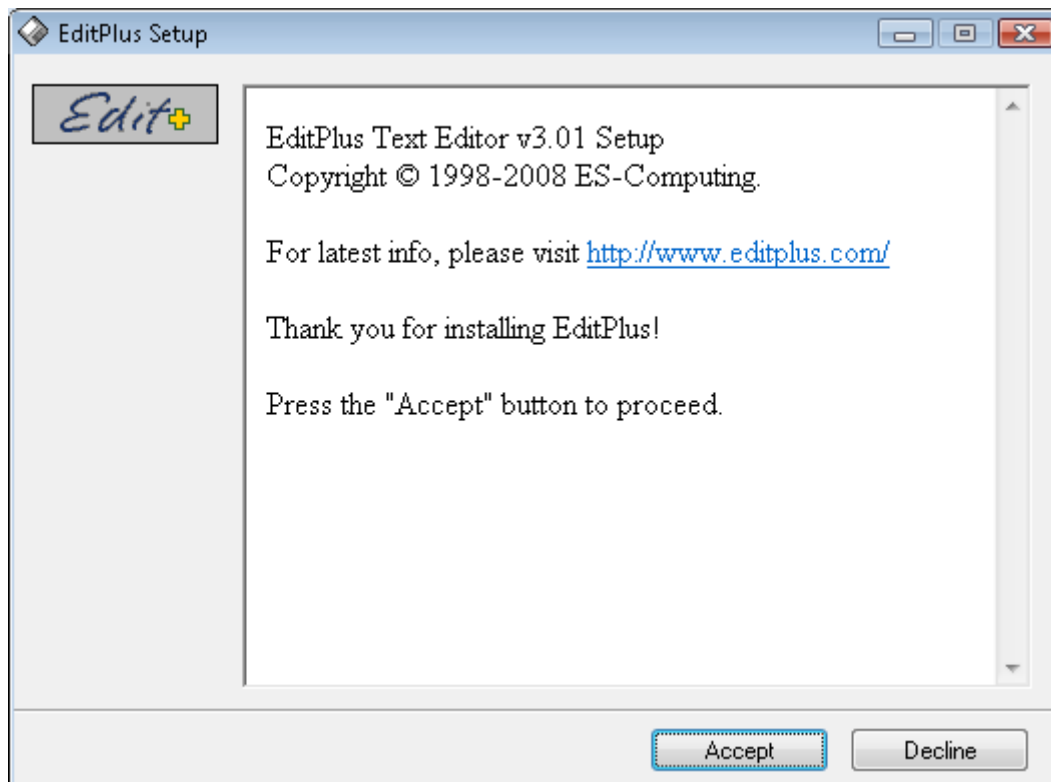
Ahora ya podemos instalar el que será el órgano común de nuestra forma de trabajar. Descomprime el EditPlus v.3.01.by.taibo.zip dentro de la carpeta del Pack.

Este es lamentablemente un software de pago, pero merece la pena que lo instales por todas las ventajas que nos da. Lee las instrucciones dentro del txt. del zip

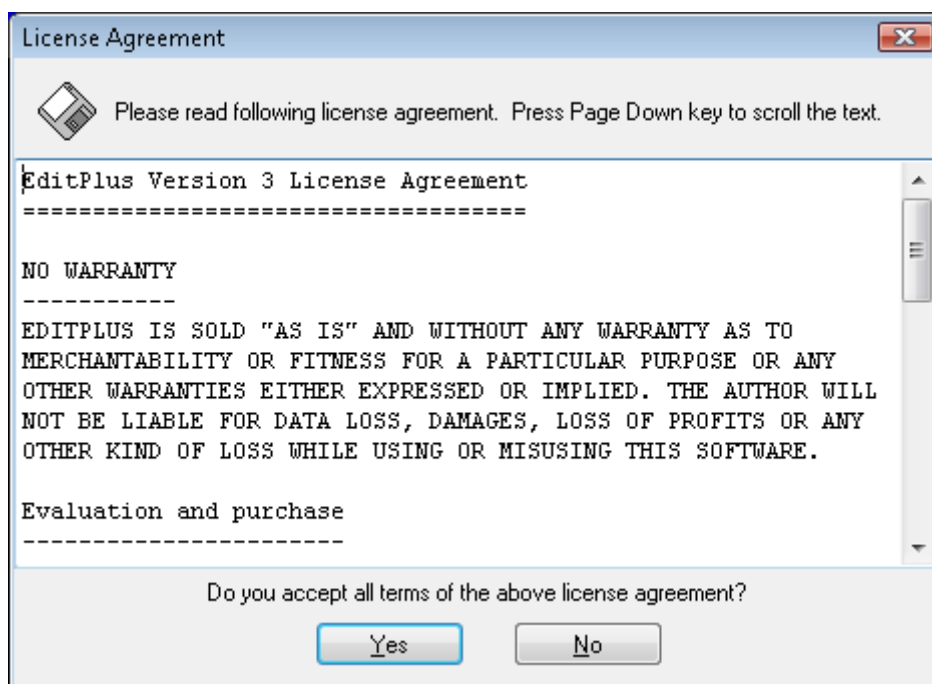
Dentro de la carpeta del Pack donde has descomprimido el EditPlus veras este icono.



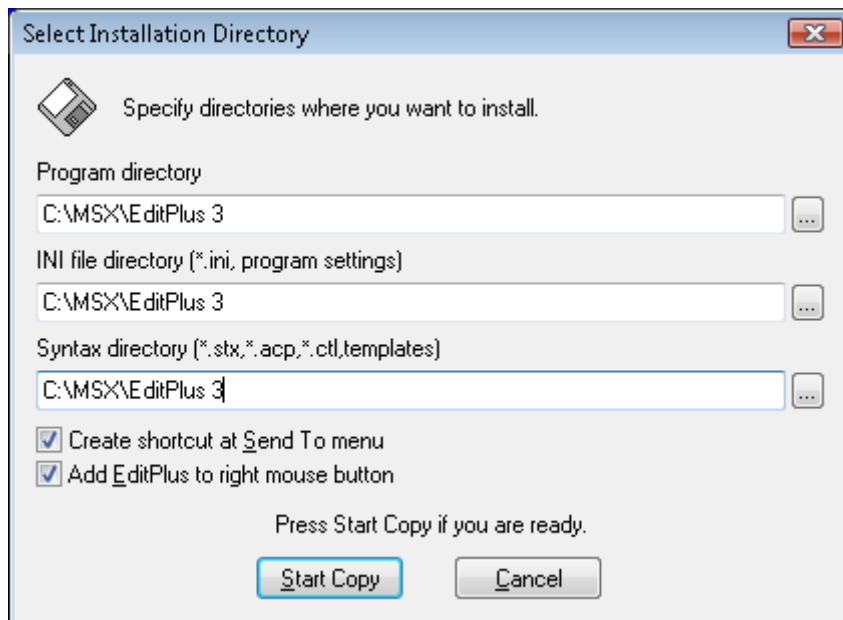
Pulsa doble clic sobre este icono para instalarlo.



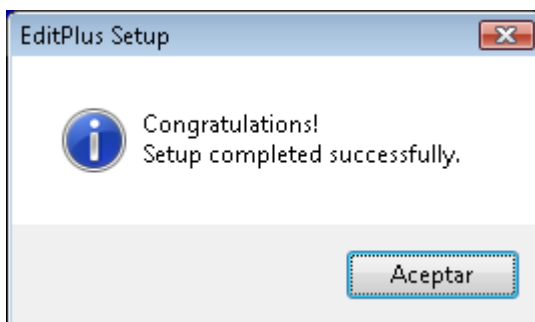
Pulsamos el botón “Accept”



Pulsa el botón “Yes” para aceptar la licencia.

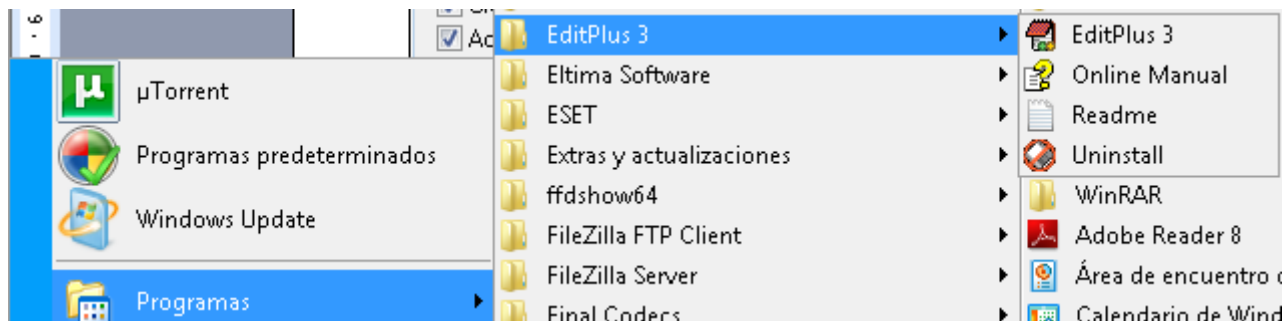


Coloca o escribe los 3 directorios apuntando a [C:\MSX\EditPlus 3](#) como ves en la imagen.

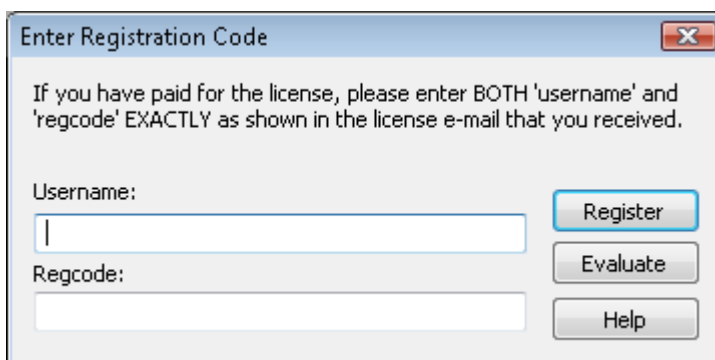


Ya lo tenemos instalado.

Pulsamos el botón “Aceptar”

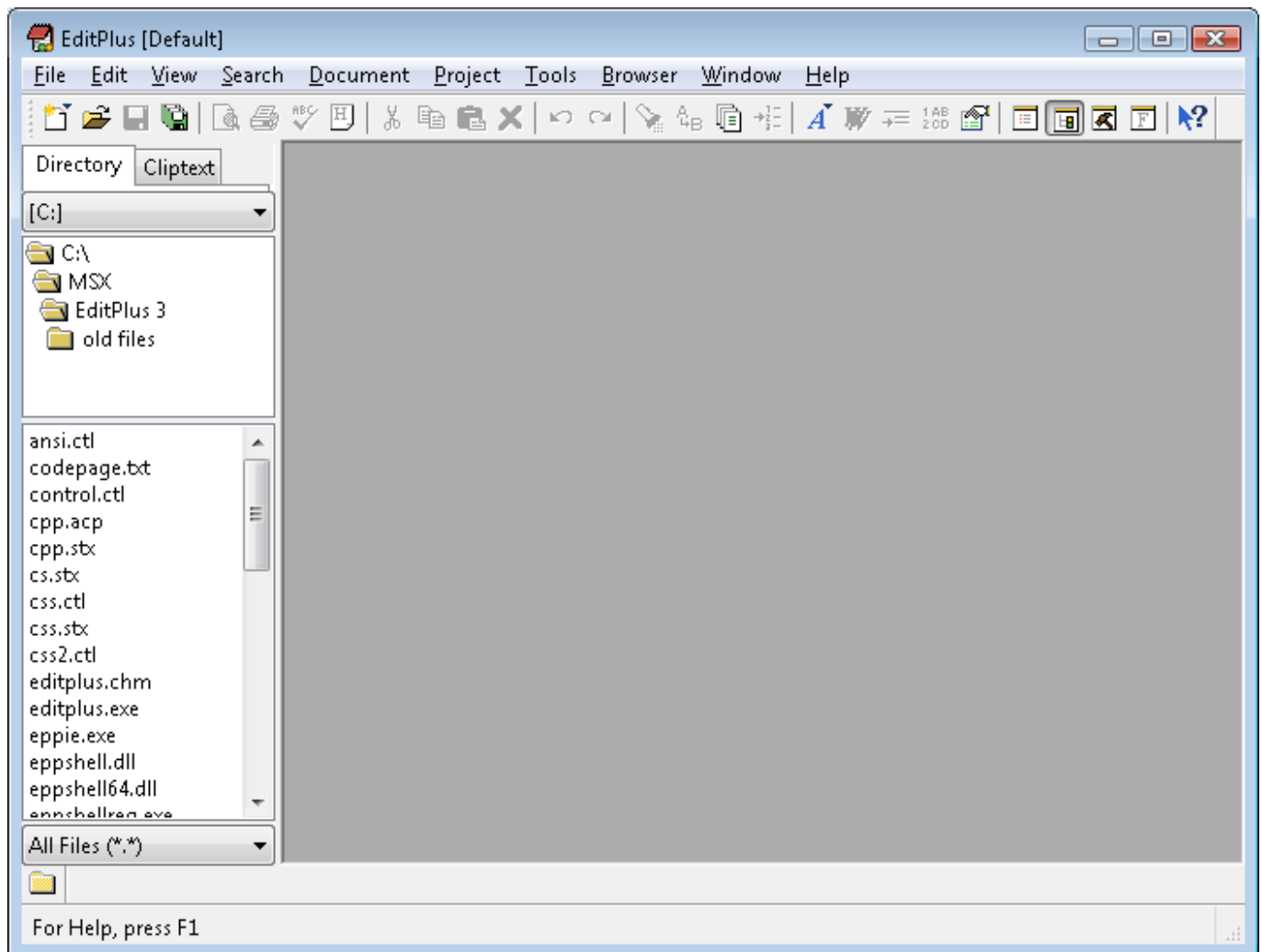


Ahora lo ejecutamos como administrador pulsa en [Programas – EditPlus 3 – EditPlus 3](#), con el botón derecho del ratón y pulsa [ejecutar como administrador](#), si lo haces en Windows vista o Windows 7 si usas Windows XP no hace falta, te recomiendo copiar este acceso al escritorio.

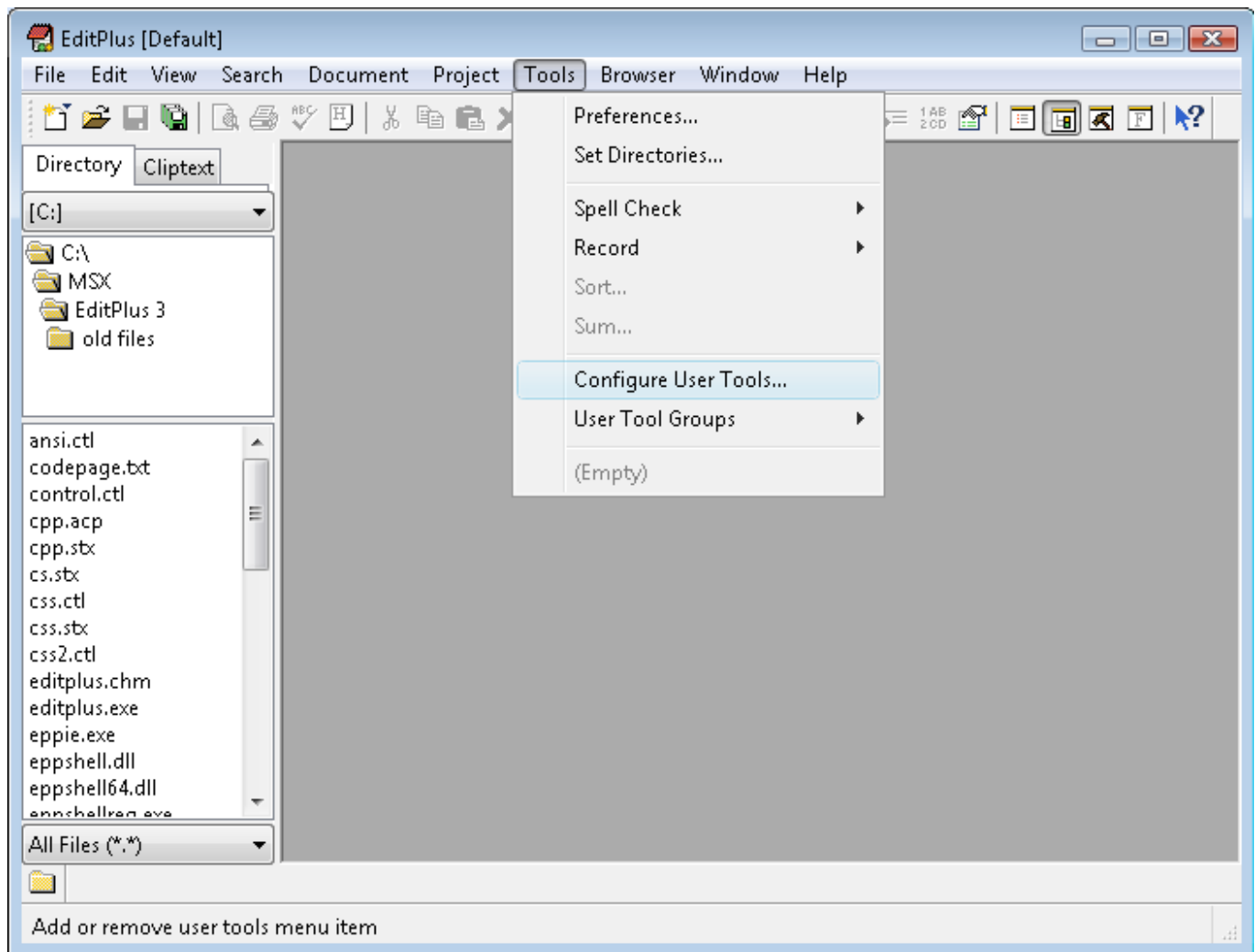


Si tienes los datos para el registro es el momento de introducirlos y Pulsar el botón “[Register](#)” si has leído el .txt te ayudara.

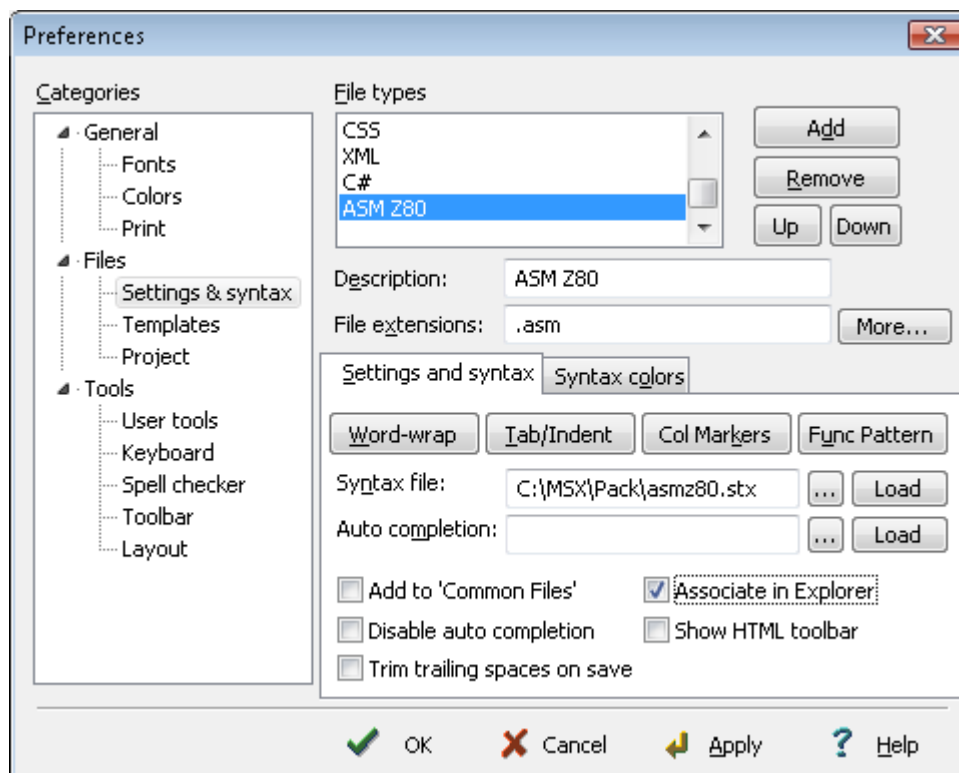
De lo contrario pulsa el botón “[Evaluate](#)” para que te dejen usar el programa durante 30 días.



Aquí tenemos nuestro editor en todo su esplendor, ahora vamos a realizar los pasos más importantes uno será, incorporarle un añadido para que nos muestre por colores la sintaxis del lenguaje en ensamblador del Zilog Z80, el fichero que vamos a incorporar al EditPlus es [asmz80.stx](#) en el Pack.



Para agregarlo vamos al menú **Tools – Configure User Tools...**



1 - Primero en la izquierda tienes que señalar la Opción de **settings & syntax** una vez señalado.

2 - Pulsamos el botón **Add** teclea en el campo **Description:** pon **ASM Z80** Y en **File extensions:** **.asm**

3 - Ahora pulsa en **Syntax file:** en el botón **...** y selecciona en la imagen que ves debajo de este cuadro el fichero **asmz80.stx** que esta en el directorio del Pack.

Marca **Associate in Explorer** Si ves la imagen como la foto de la izquierda ya puedes pulsar los botones **Apply** y **OK**

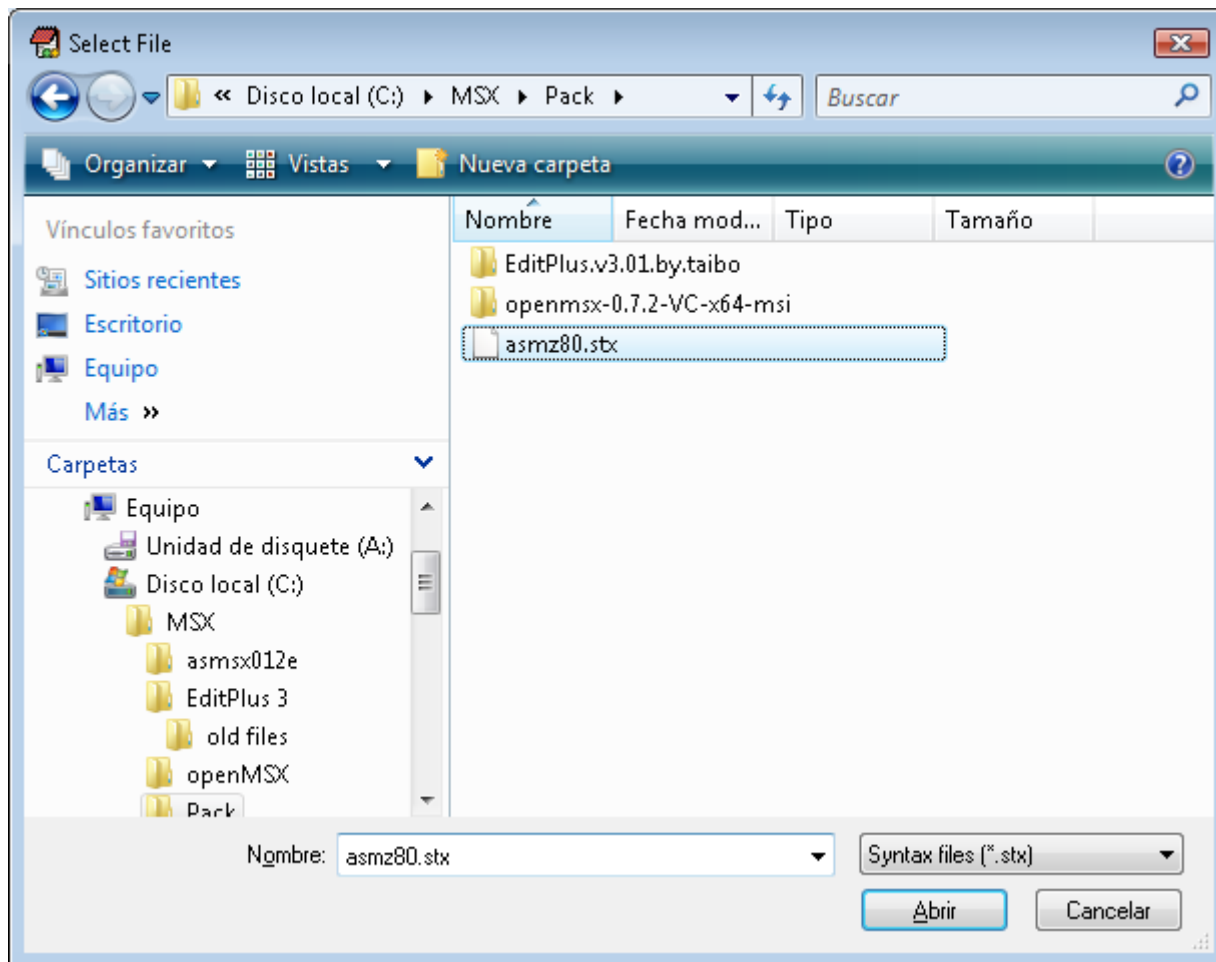
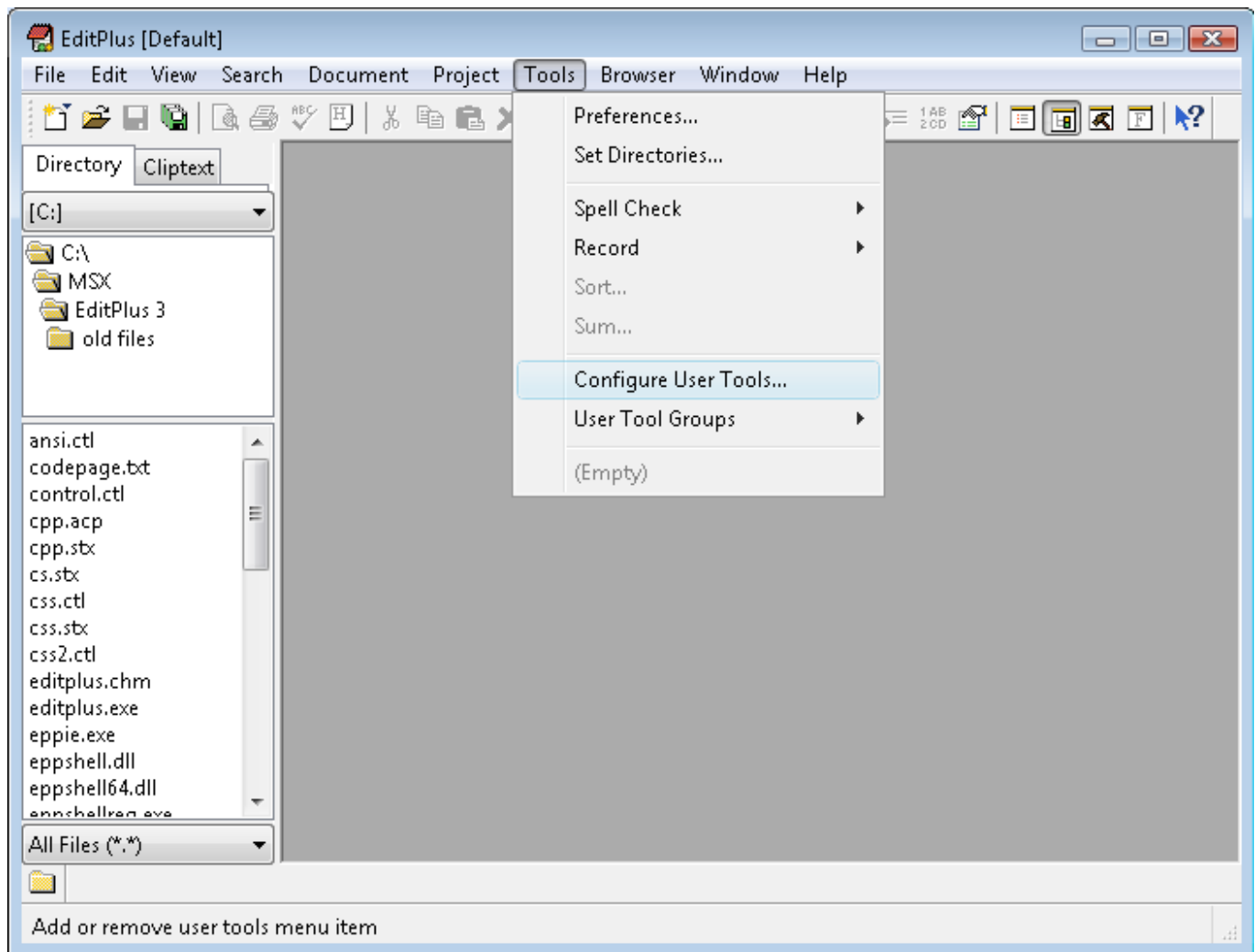
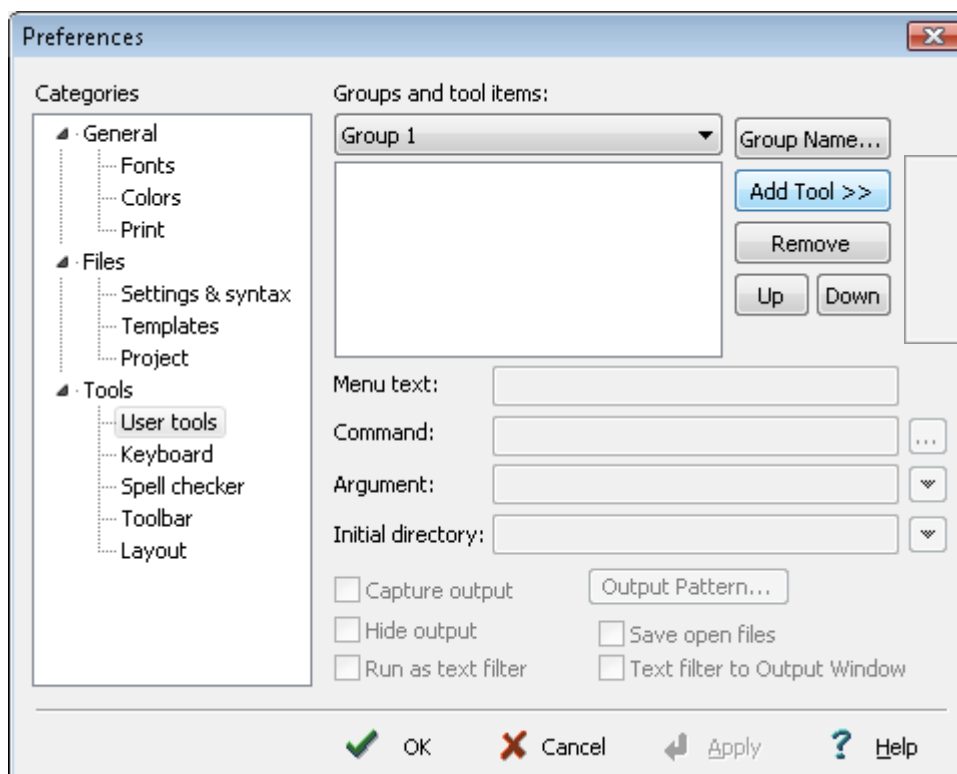


Imagen que sale cuando pulsamos en el paso 3 descrito más arriba.

Ahora vamos a crear nuestro propios botones uno será para [Compilar](#) y el otro Ejecutar el Compilado o fichero ROM en el Emulador [openMSX](#).



De nuevo pulsamos en el menú del EditPlus en **Tools – Configure User Tools...**



Primero en la Izquierda señala **User tools**

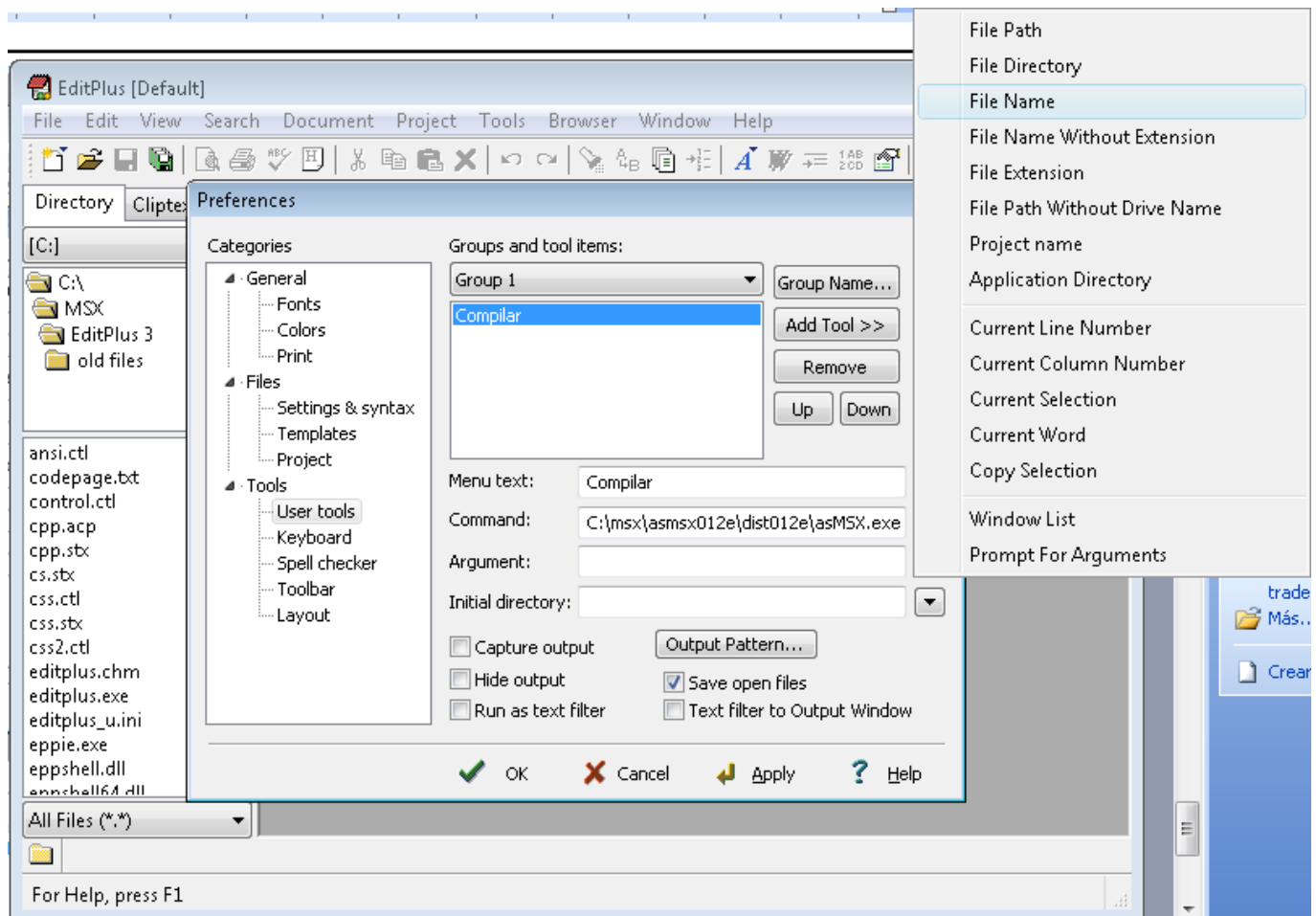
Program
Help File (*.hlp)
HTML Help file (*.chm)
Keystroke Recording

Ahora pulsa en el botón **AddTool** y en la ventana emergente elige la opción **Program**

Ahora en **Menu text:** teclea **Compilar**

Ahora pulsa en **Command:** en los 3 puntos ...

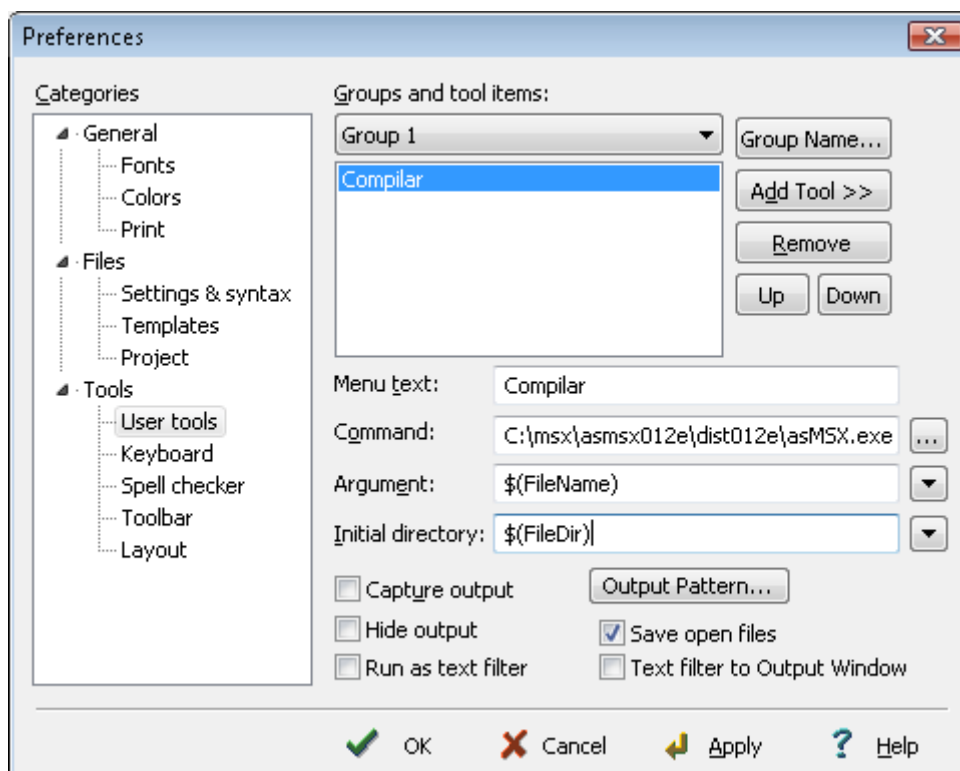
Fíjate en la imagen de abajo.



Aquí le decimos la ruta de nuestro ensamblador `c:\msx\asmsx012e\dist12e\asMSX.exe`

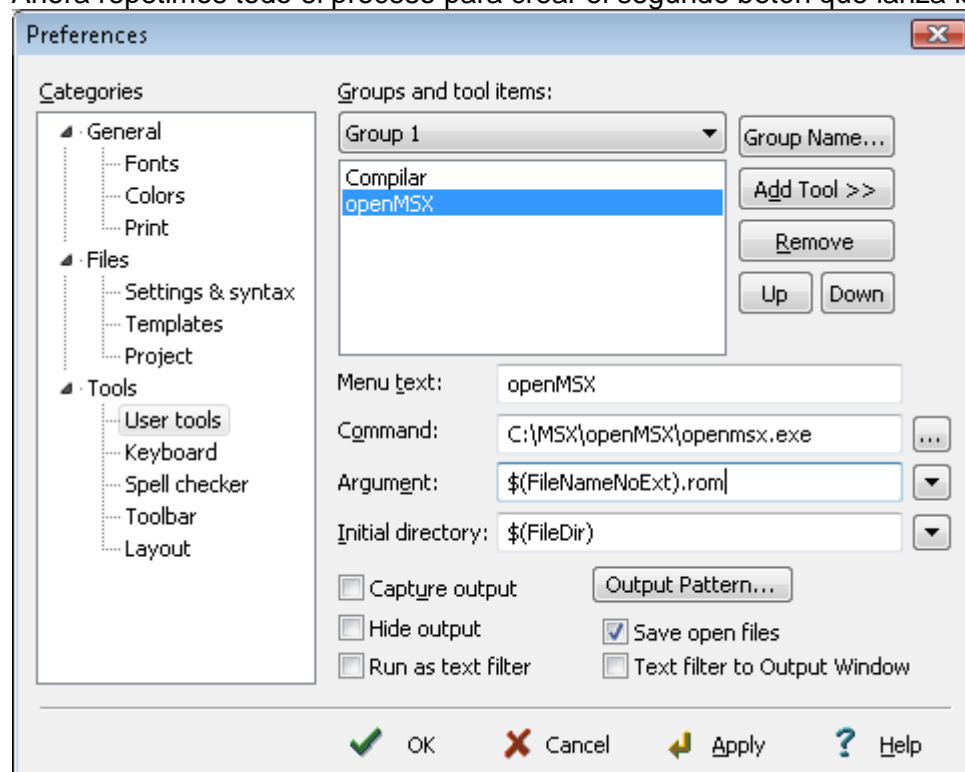
En la opción **Argument**: pulsa la flecha y selecciona **File Name**

Y en la opción **Initial directory**: pulsa la flecha y selecciona **File Directory**



Este es el resultado final que tienes que ver para el botón **Compilar**. Pulsa el botón **Apply**

Ahora repetimos todo el proceso para crear el segundo botón que lanza la ROM en el openMSX



Pulsa de nuevo en el botón **Add Tool** y en la ventana emergente elige la opción **Program**

Ahora en **Menu text:** teclea **openMSX**

Pulsa en **Command:** en los 3 puntos **...** y selecciona el directorio donde esta instalado el openMSX

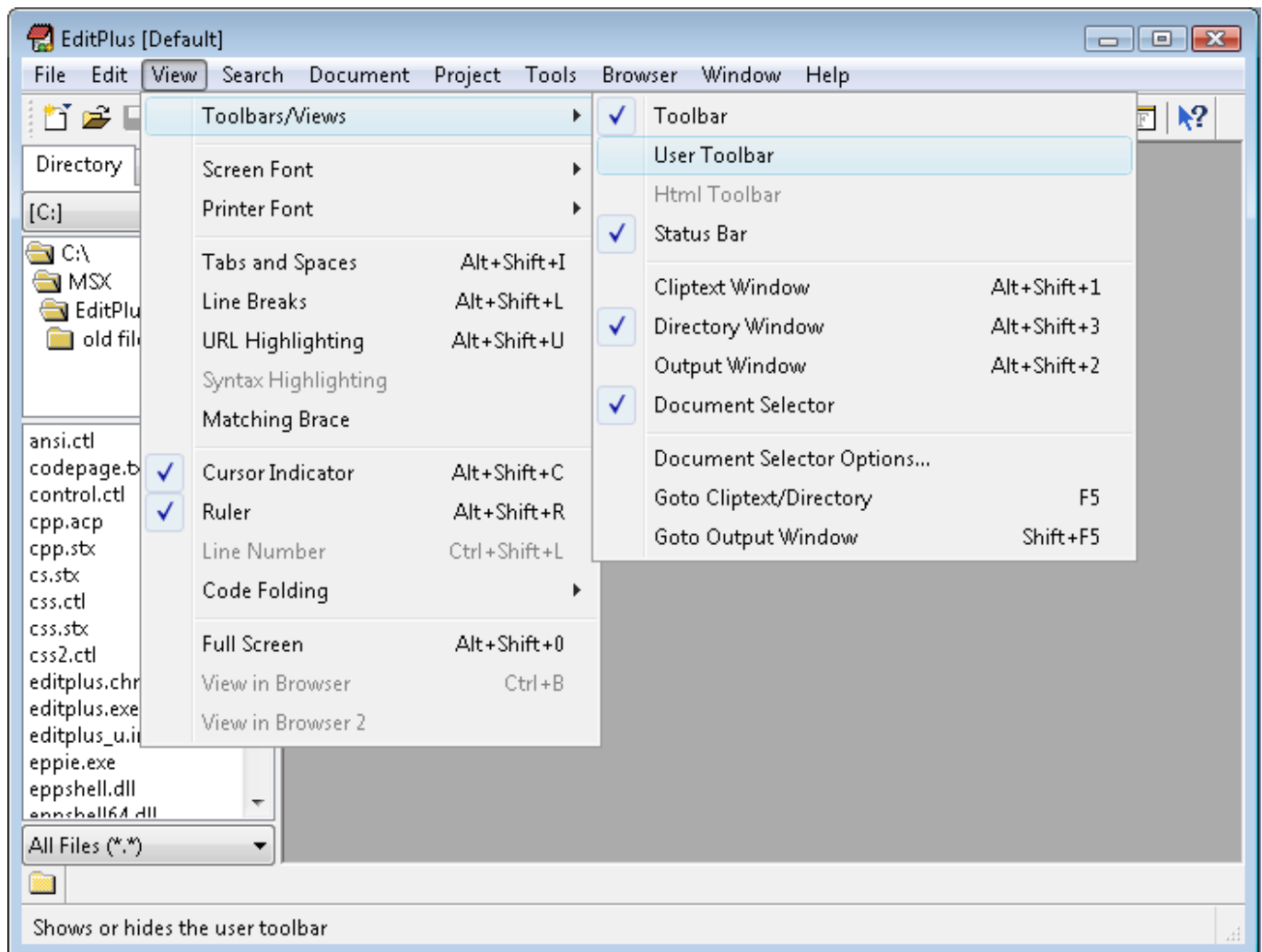
Que supongo que sera **C:\MSX\openMSX\openmsx.exe**

En la opción **Argument:** pulsa la flecha y selecciona **File Name Without Extension** y añade **.rom** al final

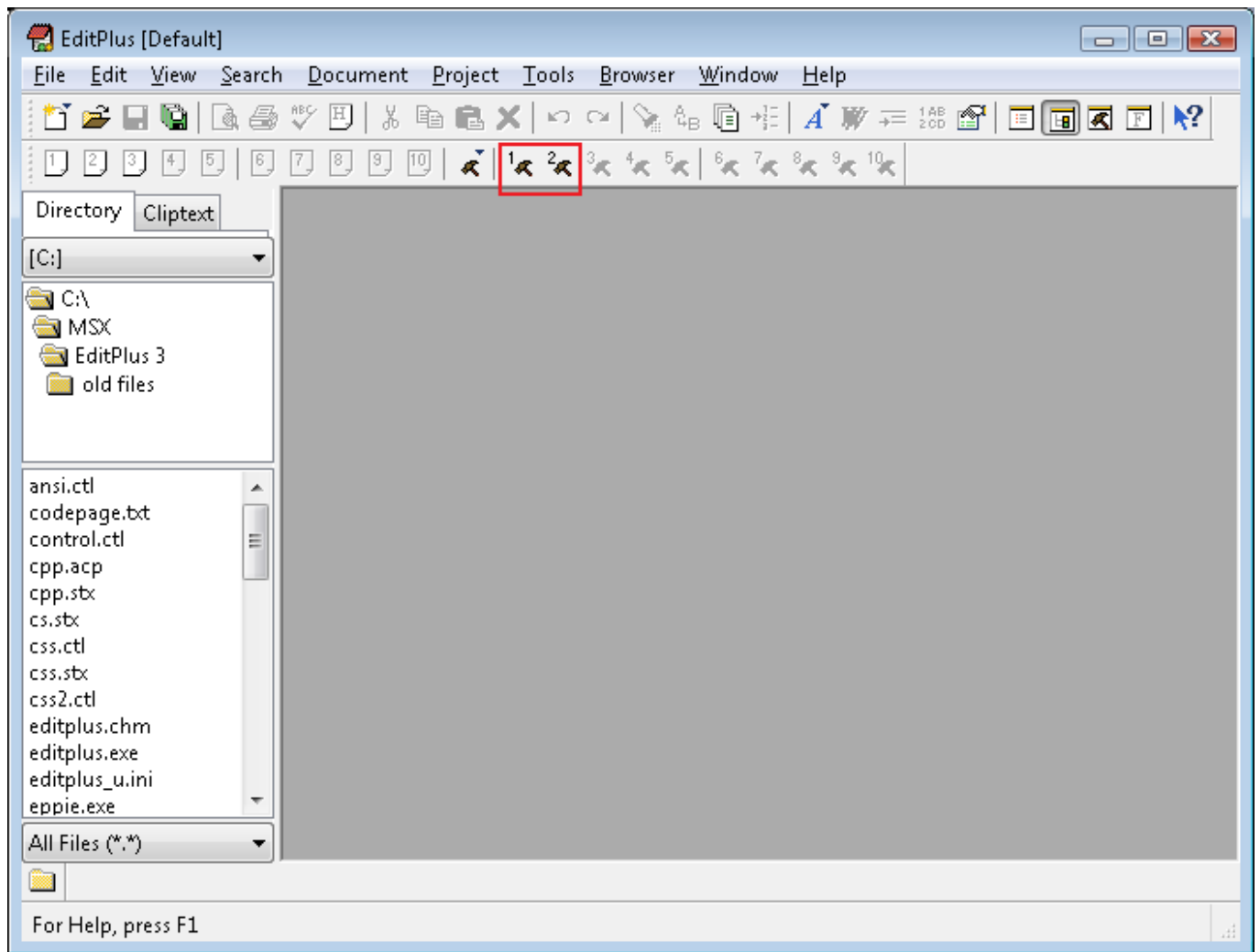
Y en la opción **Initial directory:** pulsa la flecha y selecciona **File Directory**

Fijate en la imagen de arriba de este texto, asi es como tiene que quedar la ventana.

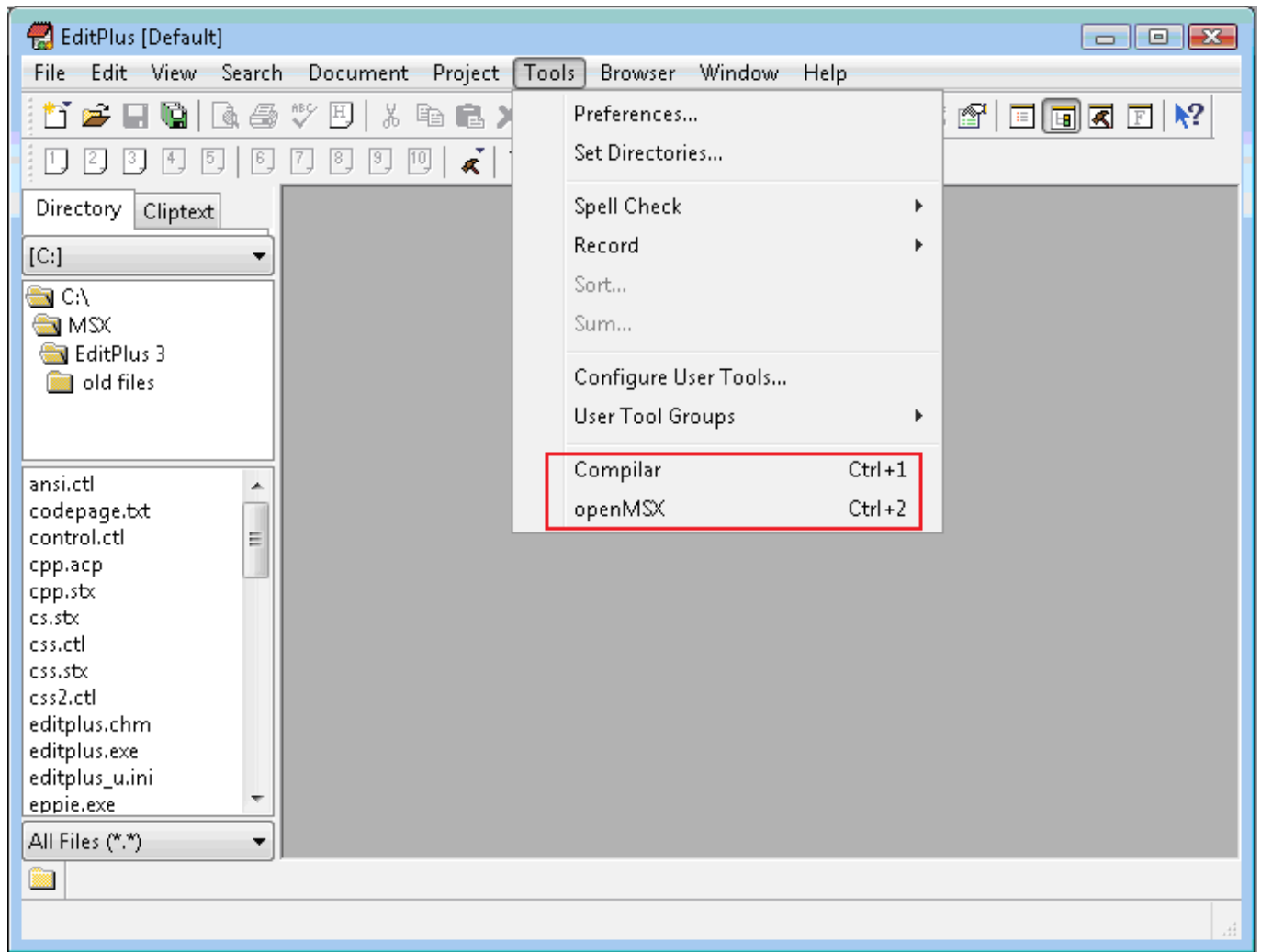
Ahora ya puedes pulsar el botón **Apply** y el botón **OK**



Ahora vamos a mostrar en la barra de iconos nuestros dos nuevos botones el [Compilar](#) y el [openMSX](#). Pulsa en menú [view – Toolbars/Views – User Toolbar](#) para que se muestre la barra User Toolbar.

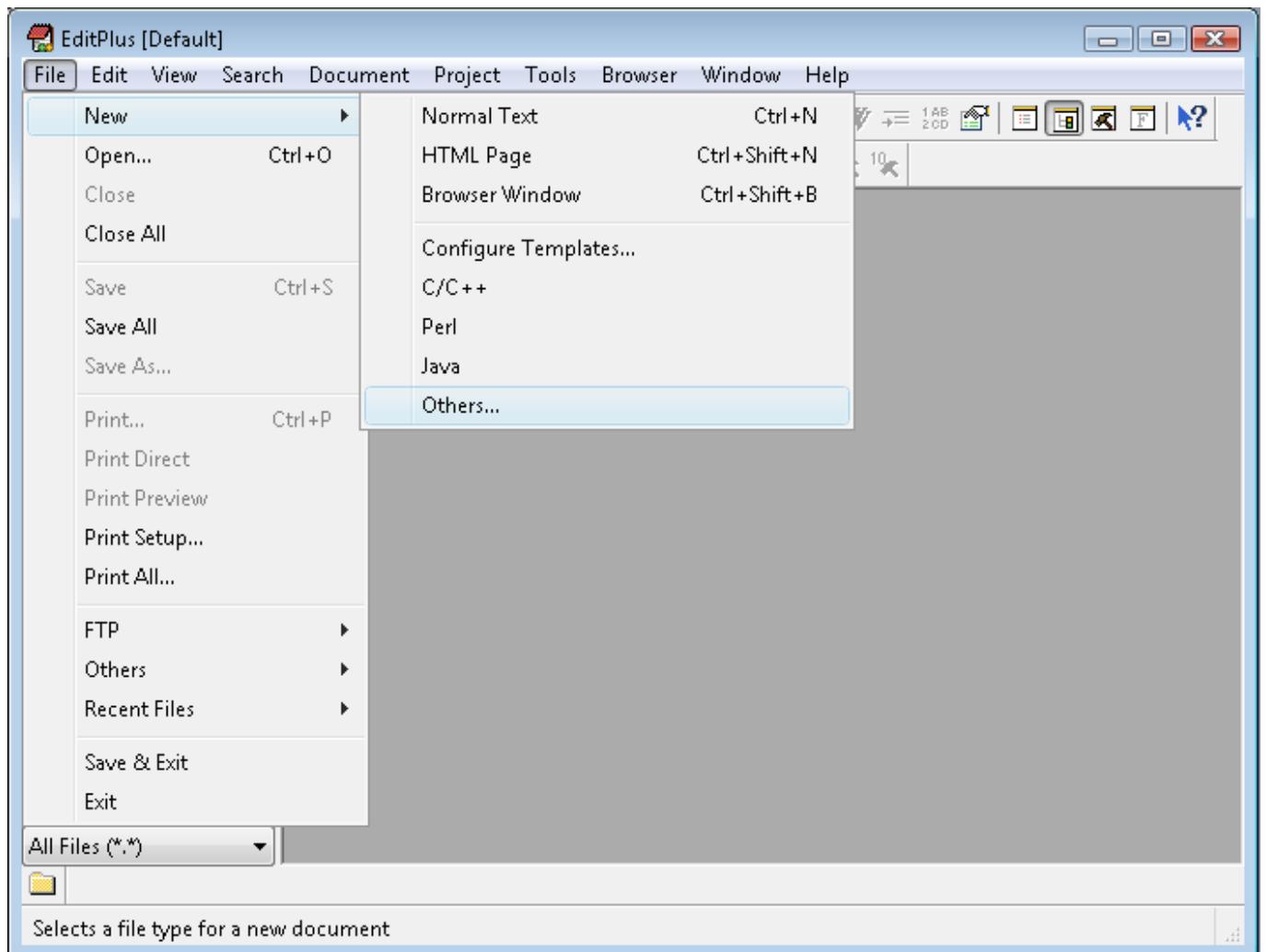


Y voila... aquí puedes ver nuestros botones [Compilar](#) y [openMSX](#) en el **recuadro rojo** martillo 1 y 2

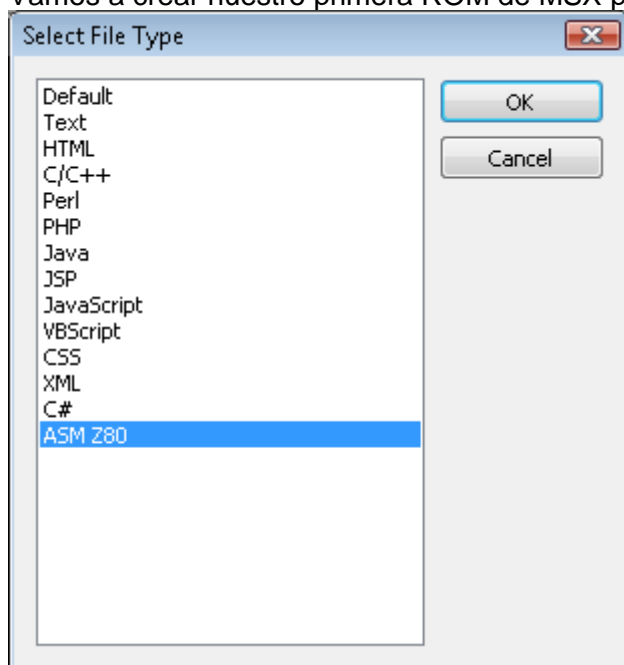


También puedes llamar a nuestros botones en [menú – Tools](#) o con [Control+1](#) y [Control+2](#)

Vamos a probar si funciona todo nuestro tinglado a la perfección, os voy a enseñar a crear vuestra primera ROM de MSX con un Hola Mundo muy sencillito usando la BIOS.



Vamos a crear nuestra primera ROM de MSX pulsa en menú [File – New – Others](#)



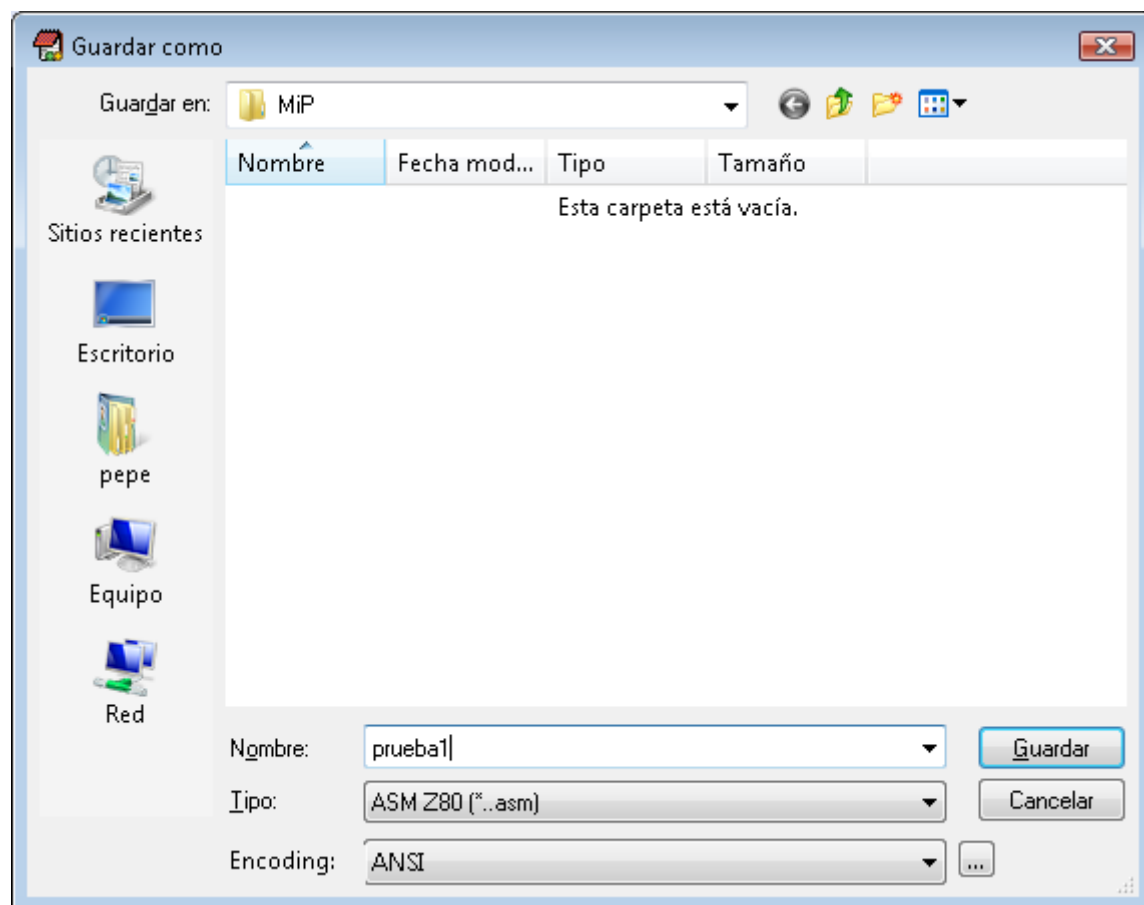
Selecciona [ASM Z80](#) y pulsa el botón [OK](#).

AHORA COPIA Y PEGA ESTE TEXTO EN TU EDITOR menos esta línea

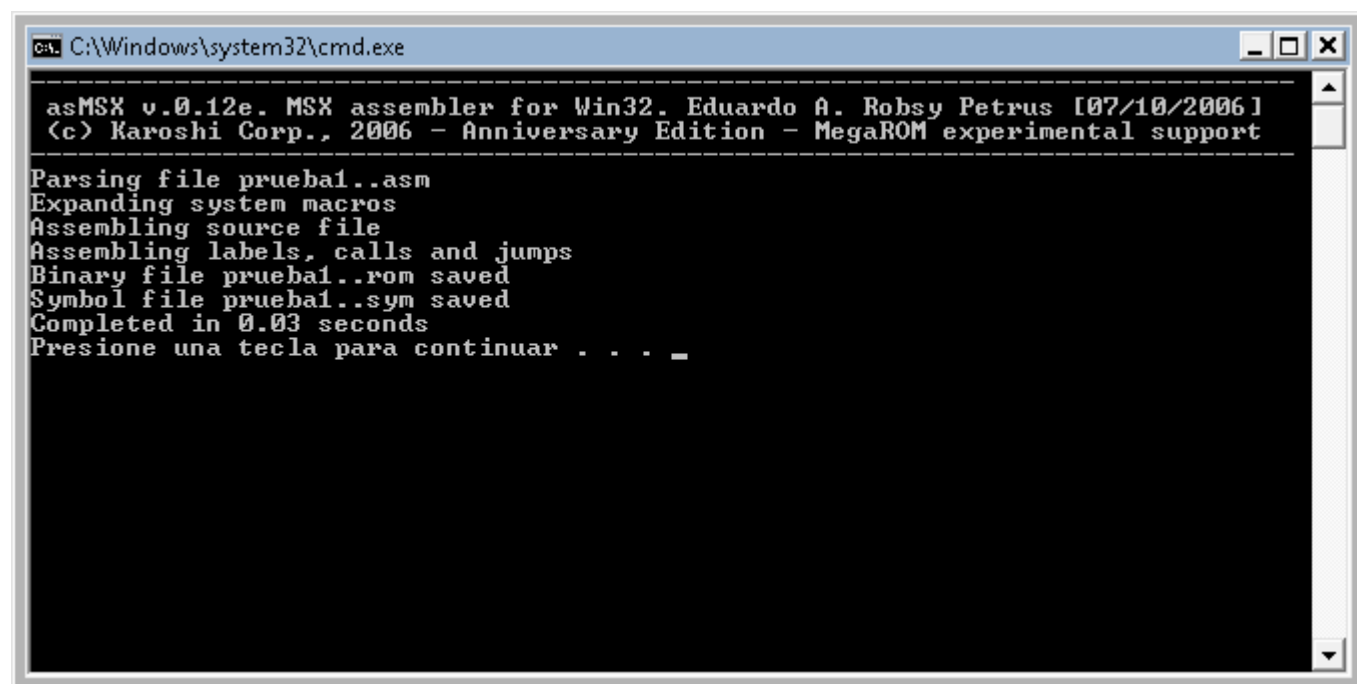
```
-----
; Nombre de nuestro programa
; Hola Mundo
-----
; DEFINIR CONTANTES
-----
; no definimos ninguna constante
;
; Variables de sistema
; FORCLR equ F3E9h ; Foreground colour
-----
; DIRECTIVAS PARA EL ENSAMBLADOR ( asMSX )
-----
.bios ; Definir Nombres de las llamadas a la BIOS
.page 2 ; Definir la dirección del código irá en 8000h
.rom ; esto es para indicar que crearemos una ROM
.start INICIO ; Inicio del Código de nuestro Programa
-----
INICIO:
; INICIO DEL PROGRAMA
-----
;
; call INIT_MODE_SC0 ; iniciar el mode de pantalla
; call IMPRI_MENSAJE ; imprimir el mensaje en pantalla
FIN:
; jp FIN ; esto es como 100 goto 100
-----
INIT_MODE_SC0:
; INICIALIZA EL MODO DE PANTALLA
-----
; BASIC: COLOR 15,0,0
; Establecer el fondo de color Negro
;
; ld hl,FORCLR
; ld [hl],15 ; Color del primer plano
; inc hl ; blanco
; ld [hl],1 ; Color de fondo
; inc hl ; negro
; ld [hl],1 ; Color del borde
; ; negro
;
; call INITXT ; set SCREEN 0
; ; call INIT32 ; set SCREEN 1
; ; call INIGRP ; set SCREEN 2
; ; call INIMLT ; set SCREEN 3
; SCREEN 0 : texto de 40 x 24 con 2 colores
; SCREEN 1 : texto de 32 x 24 con 16 colores
; SCREEN 2 : gráficos de 256 x 192 con 16 colores
; SCREEN 3 : gráficos de 64 x 48 con 16 colores
;
; ret
-----
;
; IMPRI_MENSAJE:
; RUTINA QUE IMPRIME EL TEXTO EN PANTALLA
;
;
; ld h,01 ; situamos la Columna
; ld l,01 ; y la fila para
; ; ld hl,0101h ; también podemos hacerlo de esta manera
; call POSIT ; fijar el cursor donde empezara a escribir
; ld hl,texto ; ponemos hl apuntando al texto del mensaje
bucle:
; ld a,[hl] ; cogemos el primer carácter y lo metemos en a
; or a ; comprobamos si hemos llegado al final del texto
; ret z ; y salimos de la rutina en el caso que el compare sea Zero
; call CHPUT ; escribimos ese carácter en la posición del cursor
; inc hl ; incrementamos hl para que apunte a la siguiente letra
; jr bucle ; si no hemos llegado al final continuamos escribiendo
-----
;
; texto:
; .db "Hola Mundo",0
-----
```

FINAL DEL TEXTO QUE TIENES QUE COPIAR menos esta línea

Ya estamos preparados para compilar y ejecutar el compilado en el openMSX
Así que pulsa el botón que hemos creado Compilar o botón definido 1

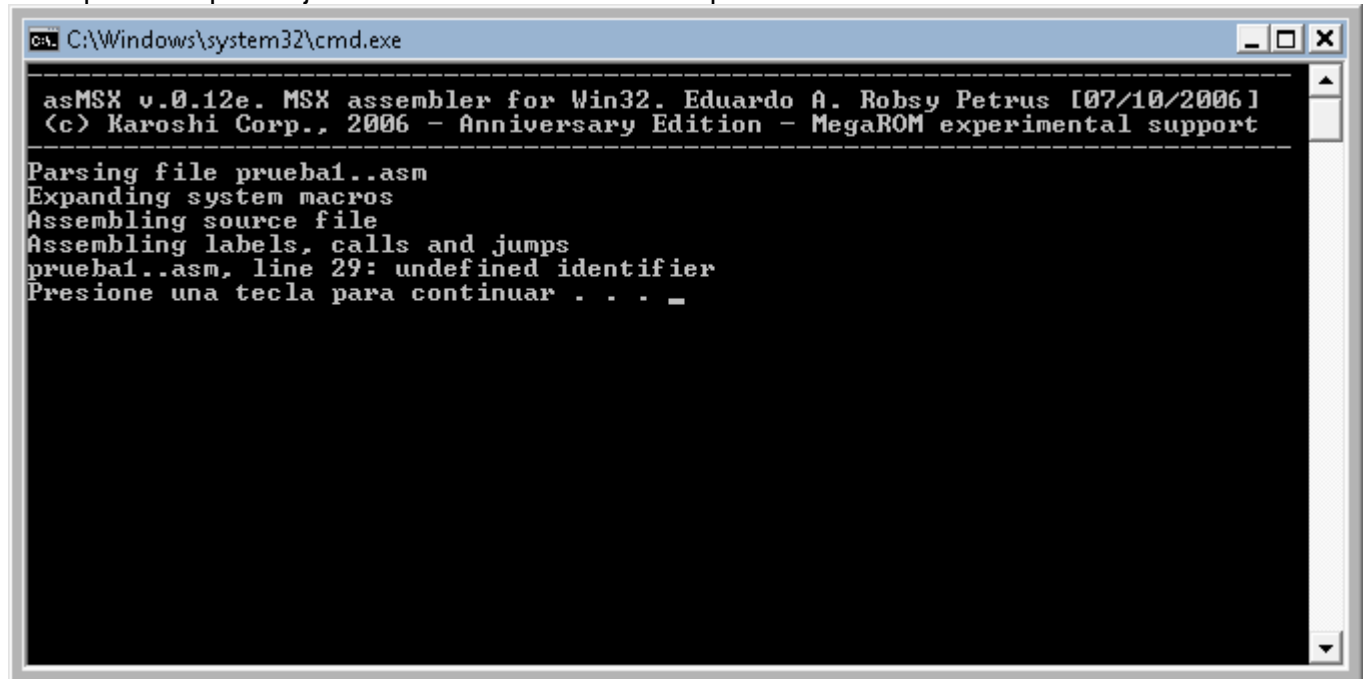


La primera vez que compilas tienes que guardar antes el fichero **.ASM** creo una carpeta llamada MiP y le pongo como Nombre: **holamundo.asm** y pulso el botón **Guardar**.



Aquí puedes ver que el **asMSX** ha compilado nuestro código en assembler sin ningún problema.

Es Importante que te fijes en esto antes de lanzar el openMSX. Cierra la ventana si todo esta OK.



```
C:\Windows\system32\cmd.exe

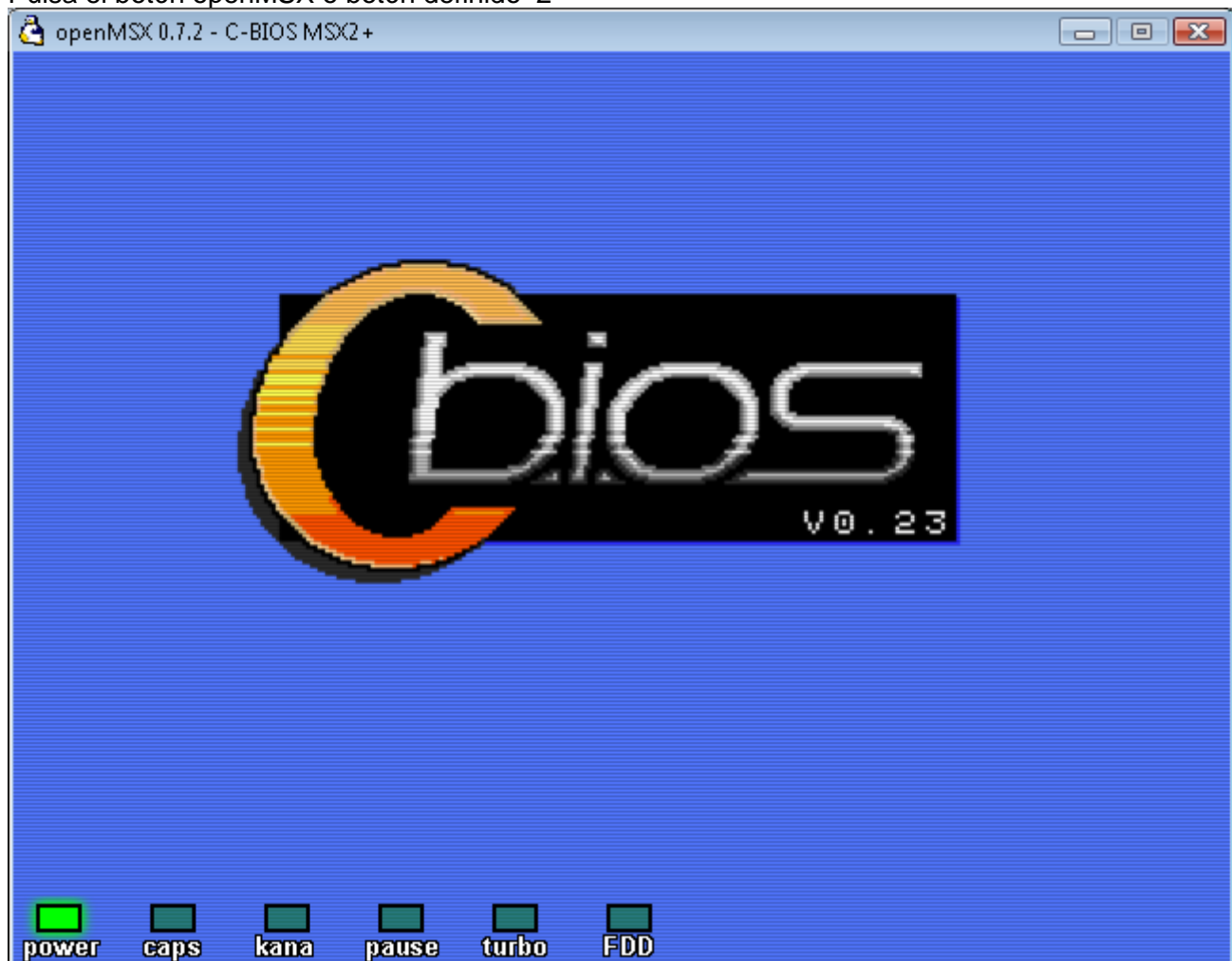
asMSX v.0.12e. MSX assembler for Win32. Eduardo A. Robsy Petrus [07/10/2006]
(c) Karoshi Corp., 2006 - Anniversary Edition - MegaROM experimental support

Parsing file prueba1.asm
Expanding system macros
Assembling source file
Assembling labels, calls and jumps
prueba1.asm, line 29: undefined identifier
Presione una tecla para continuar . . . _
```

Aquí he forzado un error de sintaxis para que veas que en caso de error, el asMSX te dice en que línea o líneas de tu código has cometido un error de sintaxis, o algo que no has definido bien, fíjate en el numero de línea y repasa con el editor de texto la línea 29 a ver que error habríamos cometido.

AHORA QUE TENEMOS EL CODIGO BIEN VAMOS A PROBARLO

Pulsa el botón openMSX o botón definido 2



Aquí puedes ver como arranca el openMSX cargando su BIOS como si fuera un MSX2 real



Y voila aquí tenemos el resultado final.

Puedes cerrar el emulador openMSX y cerrar la otra ventana negra que queda al cerrarlo.

Ya hemos creado nuestra primera ROM para MSX increíble y fácil. Ahora modifica la fila y la columna donde tiene que salir el texto en pantalla, para que veas lo fácil que es modificar compilar y lanzar.

Espero que sea de vuestro total agrado y nos vemos en el próximo tutorial. Donde explicare un poco el código de este artículo, donde se miran las rutinas de la BIOS del MSX y como se puede depurar nuestro código con el debugger del BlueMSX.

P.D. el resto de programas de Pack del MSX es para futuros capítulos.

José Vila Cuadrillero

"ES DETESTABLE ESA AVARICIA ESPIRITUAL QUE TIENEN, LOS QUE SABIENDO ALGO, NO PROCURAN LA TRANSMISION DE ESOS CONOCIMIENTOS."

Miguel de Unamuno

Escritor y Filósofo.

(Bilbao 1864 - Salamanca 1936)

TUTORIAL DE INICIACION A LA PROGRAMACION EN LENGUAJE ENSAMBLADOR PARA MSX

2ª PARTE – LA BIOS Y EL DEBUGGER

En esta parte del tutorial vamos a ver como hemos creado el programa HOLA MUNDO que vimos en la primera parte, que son las llamadas a la BIOS como podemos ver donde están y como usarlas, así como depurar el código o buscar errores usando el Debugger del BlueMSX con nuestro programa.

```
-----  
; Nombre de nuestro programa  
; Hola Mundo  
-----  
; DEFINIR CONTANTES  
-----  
; no definimos ninguna constante  
-----  
; Variables de sistema  
FORCLR equ F3E9h ; Foreground colour  
-----  
; DIRECTIVAS PARA EL ENSAMBLADOR ( asMSX )  
-----  
.bios ; Definir Nombres de las llamadas a la BIOS  
.page 2 ; Definir la dirección del código irá en 8000h  
.rom ; esto es para indicar que crearemos una ROM  
.start INICIO ; Inicio del Código de nuestro Programa
```

Vamos a comentar que es toda esta parte del código.

El parámetro **FORCLR** es una variable del sistema MSX (y os preguntareis que es esto)

Las variables del sistema son comunes para todos los ordenadores MSX y son iniciadas por la BIOS al arrancar el ordenador están disponibles para la BIOS y el BASIC y por supuesto para nosotros..

Las variables del sistema están situadas en RAM desde la dirección de memoria **F380h hasta la FFFFh**

En la posición de memoria **F3E9h** es donde el sistema almacena el color que usamos en la pantalla, compuesto por el “Color del primer plano” o color de las letras, “Color del Fondo” y “Color del Borde” la BIOS pone el color que vemos al teclear en el interprete BASIC.

Definimos en el ensamblador que **FORCLR** es igual a **F3E9h** en código “ **FORCLR equ F3E9h** “

Se me olvido poner en el **PACK** este fichero de texto con todas las variables del sistema MSX por si queréis consultarlas descargarlo de aquí. <http://www.megaupload.com/?d=Q5T1B00E>

Para futuros tutoriales ya sabéis que las variables del sistema las defino en esa sección del código.

Vamos con las directivas del Ensamblador asMSX.

Decir que todo lo aquí explicado lo podéis ver o leer en el PDF que acompaña al ensamblador cruzado asMSX. creado por el maestro **Eduardo Robsy Petrus**.

Directiva “ **.BIOS** ” esta directiva la empleamos para que el asMSX traduzca los nombres de las llamadas a la BIOS en sus posiciones de memoria.

Como vamos a ver en el código después, es más fácil para el programador acordarse o ver un nombre, en vez de una posición de la memoria.

Ejemplo de llamada a la BIOS **CALL INITXT** o **CALL 006Ch** las dos son validas.

Habilitando esta directiva podemos usar los nombres de las llamadas a la BIOS.

Directiva “ **.page 2** ” esta directiva la empleamos para que el asMSX empiece a compilar el código en la **Pagina 2** de la memoria RAM del MSX. No me voy a extender mucho ya que esto esta explicado por muchos sitios, pero a groso modo os explico un poco la RAM va desde la posición **0000h hasta la FFFFh** la **Pagina 0** es usada por la BIOS desde **0000h hasta 3FFFh** el BASIC está situado en la **Pagina 1** desde **4000h hasta 7FFFh** la **Pagina 2** es para programas y va desde **8000h hasta BFFFh** y la **Pagina 3** va desde la **C000h hasta la FFFFh**. Lo normal es utilizar los 16Kb de la pagina 2 para nuestra ROM pero si queremos crear una ROM de 32Kb usaríamos la Pagina 1 y 2 ya que el BASIC no lo necesitamos cuando programamos en lenguaje ensamblador. Incluso podríamos crear una ROM de 48Kb usando la Pagina 0, pero ojo que aquí

está la BIOS del sistema y se ha de emplear con cuidado. Ver tutorial de Ramones en la comunidad KAROSHI, la [Pagina 3](#) se emplea para nuestras variables de programa. (Ya lo explicaré.) Directiva “.rom “ esta directiva la empleamos para que el asMSX nos cree a la hora de compilar un fichero .ROM si queremos crear un .BIN usaríamos “.BASIC “ ver manual del asMSX

Directiva “.Start xxxx “ esta directiva la empleamos para que el asMSX sepa dónde empieza el código que vamos a programar, por eso después de esta directiva he puesto INICIO que como veis debajo de este texto es donde empieza nuestro programa.

```

;-----
INICIO:
; INICIO DEL PROGRAMA
;-----
    call    INIT_MODE_SC0 ; iniciar el mode de pantalla
    call    IMPRI_MENSAJE ; imprimir el mensaje en pantalla
FIN:
    jp      FIN           ; esto es como 100 goto 100

```

Esta parte del código es el bucle principal del programa, lo primero es definir la etiqueta de INICIO: que usaremos en la directiva .start después llamamos a la sub-rutina INIT_MODE_SC0 que se encarga de fijar los colores y el modo de pantalla. Después llamamos a la sub-rutina IMPRI_MENSAJE que será la encargada de sacar el texto por pantalla. Y finalmente finalizamos el programa.

```

;-----
INIT_MODE_SC0:
; INICIALIZA EL MODO DE PANTALLA
;-----
; BASIC: COLOR 15,1,1
; Establecer los colores
    Ld      hl,FORCLR      ; Variable del Sistema
    Ld      [hl],15        ; Color del primer plano 15=blanco
    inc     hl             ; FORCLR+1
    Ld      [hl],1         ; Color de fondo 1=negro
    inc     hl             ; FORCLR+2
    Ld      [hl],1         ; Color del borde 1=negro

    call    INITXT         ; set SCREEN 0

;    call    INIT32        ; set SCREEN 1
;    call    INIGRP        ; set SCREEN 2
;    call    INIMLT        ; set SCRREN 3
; SCREEN 0 : texto de 40 x 24 con 2 colores
; SCREEN 1 : texto de 32 x 24 con 16 colores
; SCREEN 2 : graficos de 256 x 192 con 16 colores
; SCREEN 3 : graficos de 64 x 48 con 16 colores
;
    ret
;-----

```

Sub-rutina INIT_MODE_SC0 : Es la encargada de poner los colores para nuestro programa y el modo de pantalla que vamos a usar la rutina es fácil de entender con los comentarios, pero básicamente es esto. Se sitúa en la posición de memoria [F3E9h](#) y colocamos un 15 en [F3EAh](#) un 1 y en [F3EBh](#) un 1 después llamamos a una rutina de la BIOS llamada [INITXT](#) que es la encargada de poner la pantalla en modo SCREEN 0 y [RET](#)ornamos o volvemos al bucle principal.

```

;-----
IMPRI_MENSAJE:
; RUTINA QUE IMPRIME EL TEXTO EN PANTALLA
;-----
;
    Ld      h,01           ; situamos la Columna de la pantalla
    Ld      l,01           ; situamos la fila de la pantalla
;    Ld      hl,0101        ; también podemos hacerlo de esta manera
    Call    POSIT          ; BIOS fijar el cursor donde empezara a escribir
    Ld      hl,mensaje     ; ponemos hl apuntando al texto del mensaje

bucle:
    Ld      a,[hl]         ; cogemos el primer carácter y lo metemos en A
    or      l              ; comprobamos si hemos llegado al final del texto
    ret     z              ; y salimos de la rutina en el caso que el compare sea Zero
    call    CHPUT          ; BIOS escribimos ese carácter en la posición del cursor
    inc     hl             ; incrementamos hl para que apunte a la siguiente letra
    jr     bucle           ; vamos a repetir de nuevo el proceso
;-----
mensaje:
    .db     "Hola Mundo",0

```

Sub-rutina **IMPRI_MENSAJE** es la encargada de escribir el texto en la pantalla usando la BIOS, lo primero es situar la columna y la fila donde vamos a empezar a escribir el texto, esto sería el equivalente al **BASIC** a **LOCATE 0,0** aquí voy a extenderme un poquito para explicaros el uso de las rutinas de la BIOS, como programador debo deciros que precisamente usar la BIOS no es la manera más rápida de hacer las cosas, ya que por código se puede optimizar mejor la velocidad pero por el contrario también ayuda a ahorrar espacio para nuestro código, así que yo personalmente utilizo aquellas rutinas de la BIOS en la que no requiero velocidad facilitándome y ahorrando espacio en la tarea como programador. En el **Pack-MSX** de la primera parte veréis que hay un fichero comprimido llamado RB4BIOS.zip este ZIP contiene un PDF con todas las rutinas de la BIOS, así como los parámetros que hay que pasarle, los parámetros que nos devuelve al llamarla y los registros del z80 que serán modificados, veamos la llamada a la **BIOS POSIT** “ **call POSIT** ” abrimos el documento y vemos de una vez todas las llamadas a BIOS en una pequeña pero muy interesante tabla.

00B7H	BREAKX	046FH	Check CTRL-STOP key directly
00BAH	ISCNTC	03FBH	Check CRTL-STOP key
00BDH	CKCNTC	10F9H	Check CTRL-STOP key
00C0H	BEEP	1113H	Go beep
00C3H	CLS	0848H	Clear screen
00C6H	POSIT	088EH	Set cursor position
00C9H	FNKSB	0B26H	Check if function key display on
00CCH	ERAFNK	0B15H	Erase function key display
00CFH	DSPFNK	0B2BH	Display function keys
00D2H	TOTEXT	083BH	Return VDP to text mode
00D5H	GTSTCK	11EEH	Get joystick status

Aquí puedes ver de un solo vistazo el nombre de la llamada a la BIOS y una breve descripción de la función que realiza en ingles “ **Set cursor position** ” o en español “ **Fijar la posición del cursor** “. Veamos cómo funciona esta rutina de la BIOS. Busca en el PDF el texto POSIT hasta que veas esto.

```
Address... 088EH
Name..... POSIT
Entry..... H=Column, L=Row
Exit..... None
Modifies.. AF, EI
```

Standard routine to set the cursor coordinates. The row and column coordinates are sent to the OUTDO standard routine as the parameters in an ESC,"Y",Row+1FH, Column+1FH sequence. Note that the BIOS home position has coordinates of 1,1 rather than the 0,0 used by the BASIC Interpreter.

Los parámetros que nos hace falta para llamar a la rutina lo ves en **Entry..** los parámetros de salida los ves en **Exit..** y los registros del z80 modificados lo ves en **Modifies..**

Veamos en **Entry..** H=Columna y L=Fila antes de llamar a esta rutina en el registro HL tenemos que poner la columna y la Fila y llamar a la rutina. (Esto son los parámetros de entrada)

Veamos en **Exit...** No contiene nada o no devuelve nada. (Esto son los parámetros de salida)

Veamos en **Modifies...** Nos modifica los registros AF y EI ya sabemos que al llamar a esta rutina si estamos usando el registro AF tendremos que guardarlo en la pila antes de llamar a esta rutina ya que quedara modificado. Un ejemplo sería **PUSH AF , CALL POSIT , POP AF** esto es todo lo que tenéis que fijaros en las llamadas a las Rutinas de la BIOS.

De ahí viene que antes de llamar a la rutina POSIT en mi programa ponga **ld h,01** y **ld l,01** para que lo veáis más claramente pero yo pondría en el código **ld hl,0101h** con esto le decimos lo mismo pero ocupa menos que si lo usamos por separado (Ahora eso si la numeración la pongo en Hexadecimal.)

También es importante leer las explicaciones de todo lo que hace la rutina de la BIOS en todo el texto que hay debajo de los parámetros de llamada. (Ya que aquí nos explica el funcionamiento)

Seguimos con mi programa.

```

    Ld      hl,mensaje      ; ponemos HL apuntando al texto del mensaje
bucle:
    ld      a,[hl]          ; cogemos el primer carácter y lo metemos en A
    or      l               ; comprobamos si hemos llegado al final del texto
    ret     z               ; y salimos de la rutina en el caso que el compare sea Zero
    call    CHPUT           ; BIOS - escribimos ese carácter en la posición del cursor
    inc     hl              ; incrementamos HL para que apunte a la siguiente letra
    jr      bucle           ; vamos a repetir de nuevo el proceso
;-----
mensaje:
    .db     "Hola Mundo",0
```

Esta es la parte del código que ira pintando letra por letra nuestro mensaje en la coordenadas de la pantalla que previamente le hemos indicado con la llamada a la BIOS en **POSIT**.

Definimos una posición en memoria llamada mensaje que contiene la cadena de texto que queremos imprimir, y la finalizamos con un valor **0** que nos indicara el final del texto y a continuación creamos un bucle que realiza la siguiente función. HL apunta a la dirección en RAM donde está el primer carácter de texto o letra que queremos pintar, metemos en el registro **A** la primera letra comprobamos si es **0** para saber si hemos llegado al final del texto y dejar de pintar letras si el resultado del **OR A** es igual a cero el Flag de **Zero** se activa y se produce el **RET Z** de lo contrario seguimos con una llamada a la BIOS en este caso **CHPUT** ahora si consultas de nuevo el PDF con las llamadas a la BIOS veras que esta rutina solo funciona en modo de pantalla de TEXTO, y la tienes que llamar con la letra a pintar en el registro **A** la columna del cursor se incrementa en uno para pintar la siguiente letra a la derecha de la que acabamos de pintar.

Siguiendo con el código lo que hará es pintar la letra **H** en la coordenada 1,1 incrementamos en uno **HL** para que apunte a la letra **o** y saltamos de nuevo a bucle repitiendo todo el proceso hasta que se encuentre con el carácter **0** saliendo de la rutina y volviendo al bucle principal.

Esto sería todo lo que hace nuestro primer programa, como se emplean las rutinas de la BIOS donde se miran y como se usan, y que parámetros de entrada y de salida así como registros modificados.

Te recomiendo como programador que cuando tú crees rutinas o sub-rutinas en tu código escribas comentarios y en la cabecera de la rutina describas lo que hace que parámetros de entrada o de salida necesitas y que registros modificas porque un programa como un videojuego se realiza en muchos meses y dependiendo de tu tiempo libre en años y cuanto mejor dejes todo comentado menos problemas tendrás para recordarlo con el paso del tiempo.

Ejemplo.

```

;-----
IMPRI_MENSAJE:
; RUTINA QUE IMPRIME EL TEXTO EN PANTALLA
;-----
;
    ld      h,01            ; situamos la Columna de la pantalla
    .....
    .....
```

Ejemplo correcto:

```

;-----
IMPRI_MENSAJE:
; RUTINA QUE IMPRIME EL TEXTO EN PANTALLA
; EN LA COLUMNA Y FILA INDICADA
; Entrada :      ninguna
; Salida :      ninguna
; Registros Modificados:  AF y HL
;-----
;
    ld      h,01            ; situamos la Columna de la pantalla
    .....
    .....
```

Esto es una práctica muy buena que no debes de perder nunca aunque con ello te lleve o pierdas más tiempo realizándola. Al final siempre será tiempo que has ganado. (y sino ya me lo contarás jejeje.)

Podrías crearte rutinas al estilo de la BIOS de la siguiente manera.

La sub-rutina seria PRINTXT el resto iría en el bucle principal experimenta con el código que te doy y realiza las pruebas y cambios que estimes oportuno.

```
-----
INICIO:
; INICIO DEL PROGRAMA
-----
        call    INIT_MODE_SC0    ; iniciar el modo de pantalla

        ld      hl,texto1        ; Dirección Mensaje 1
        call    PRINTXT          ; rutina encargada de Imprimir el texto
        ld      hl,texto2        ; Dirección Mensaje 2
        call    PRINTXT          ; rutina encargada de Imprimir el texto

FIN:
        jp      FIN              ; esto es como 100 goto 100

-----
PRINTXT:
; RUTINA QUE IMPRIME EL TEXTO EN PANTALLA
; EN LA COLUMNA Y FILA DONDE ESTA EL CURSOR
; Entrada :      HL = Dir. de la cadena de texto a pintar
; Salida :      ninguna
; Registros Modificados:  AF y DE
-----
;
        ld      d,[hl]          ; metemos en D la columna
        inc     hl              ; hl=hl+1
        ld      e,[hl]          ; metemos en E la Fila
        inc     hl              ; hl=hl+1
        ex      de,hl           ; DE a HL y HL a DE
        call    POSIT            ; fijamos las coordenadas del cursor

@@bucle:
        ld      a,[de]          ; cogemos el primer carácter y lo metemos en A
        or      a               ; comprobamos si hemos llegado al final del texto
        ret     z               ; y salimos de la rutina en el caso que el compare sea Zero
        call    CHPUT           ; BIOS escribimos ese carácter en la posición del cursor
        inc     de              ; incrementamos DE para que apunte a la siguiente letra
        jr      @@bucle         ; vamos a repetir de nuevo el proceso
-----
texto1:
        .db      01,01,"Hola Mundo",0
texto2:
        .db      01,02,"Este es mi segundo programa",0
```

Con esto tendríamos nuestro segundo programa, espero que todo lo explicado lo hayáis entendido y asimilado sino es así repásalo tantas veces como te haga falta para entenderlo.

Si utilizamos etiquetas condicionales en el [asMSX](#) como **@@bucle**: podremos emplear este nombre en otra rutina que tenga un bucle, si ponemos **bucle**: solo lo podremos emplear una vez en esa rutina.

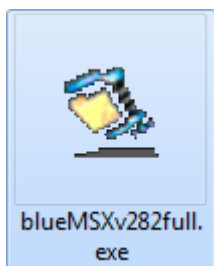
Vamos con el DEBUGGER o DESENSAMBLADOR del blueMSX desensamblar es lo contrario a ensamblar, el asMSX transforma nuestro código ensamblador en código maquina que entiende nuestro ordenador, el Debugger del blueMSX realiza lo contrario pasa el código maquina a ensamblador, el Debugger es lo que nos va a permitir tracear o ejecutar el código paso a paso y ver cómo actúa el código con el procesador Z80 como se modifican los registros los flags y la memoria o cualquier otro dispositivo que este asociado al Debugger. (Veamos como buscar errores en nuestro programa)

Para fijar un punto de Interrupción en nuestro programa hemos de incluir la directiva **.BREAK** en el punto del código donde queremos que el Debugger del blueMSX pare la ejecución de nuestro programa.

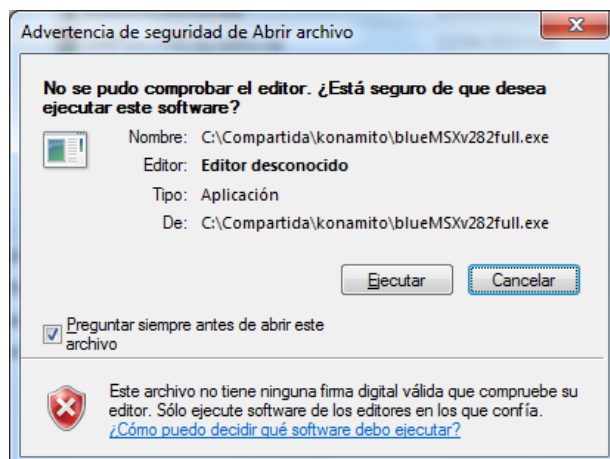
```
-----
INICIO:
; INICIO DEL PROGRAMA
-----
        .break                ; directiva del asMSX para crear un punto de interrupción aquí mismo.
        call    INIT_MODE_SC0    ; iniciar el modo de pantalla
        call    IMPRI_MENSAJE    ; imprimir el mensaje en pantalla

FIN:
        jp      FIN              ; esto es como 100 goto 100
```

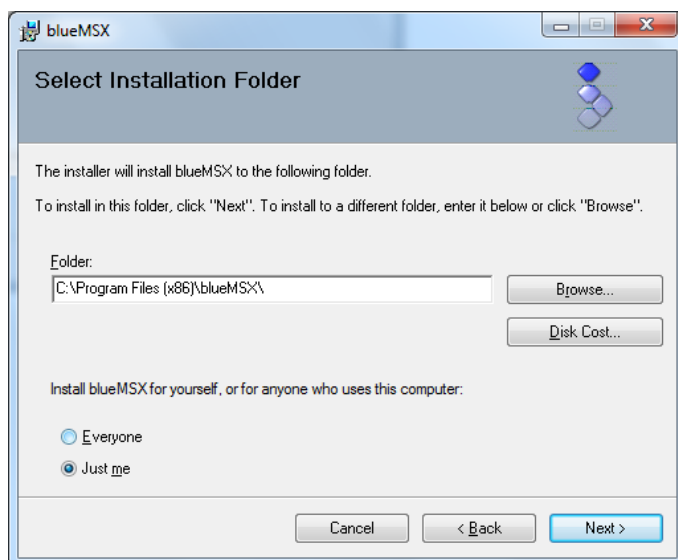
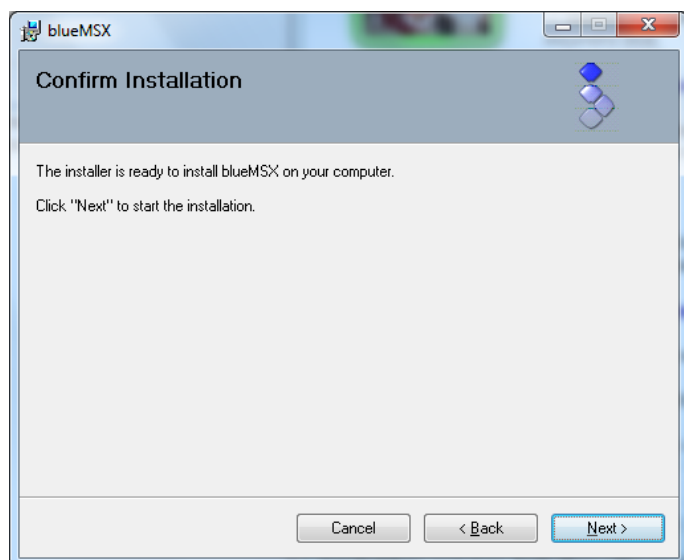
Como puedes ver he puesto el punto de interrupción en el bucle principal del programa o donde la directiva **.START** tiene el punto de entrada al programa ya que quiero tracear mi programa desde el principio pero la directiva **.break** podemos ponerla en cualquier parte del código que queramos. En el **PACK** tienes la última versión del BlueMSX en el momento de este artículo blueMSXv282full.exe



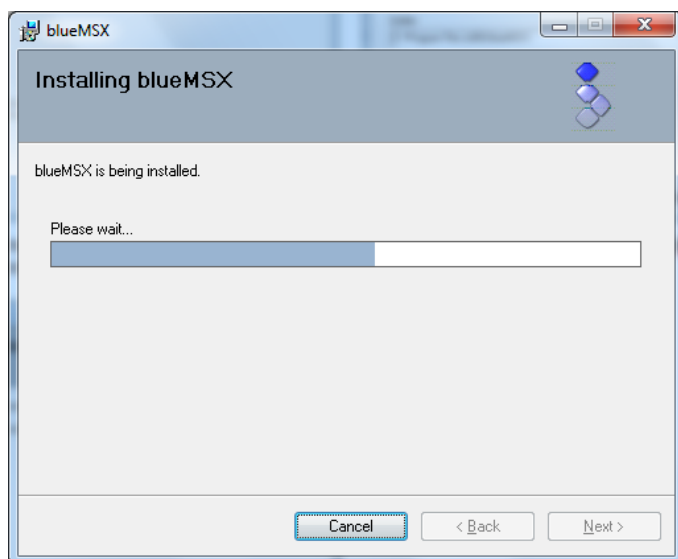
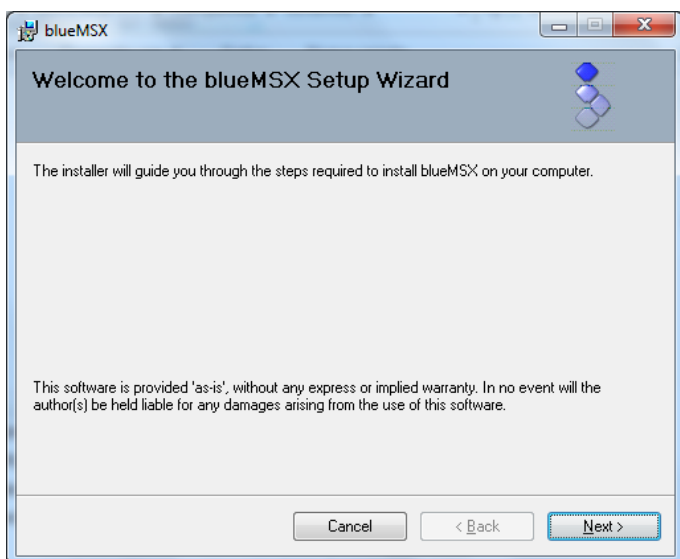
Pulsa doble clic sobre el icono del blueMSXv282full.exe para que comience la Instalación.



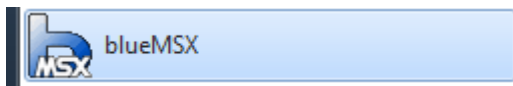
Pulsa el botón Ejecutar.



Comienza la instalación Pulsamos Next – Seleccionamos el directorio donde instalar el BlueMSX yo dejo el directorio por defecto que me sugiere el programa.



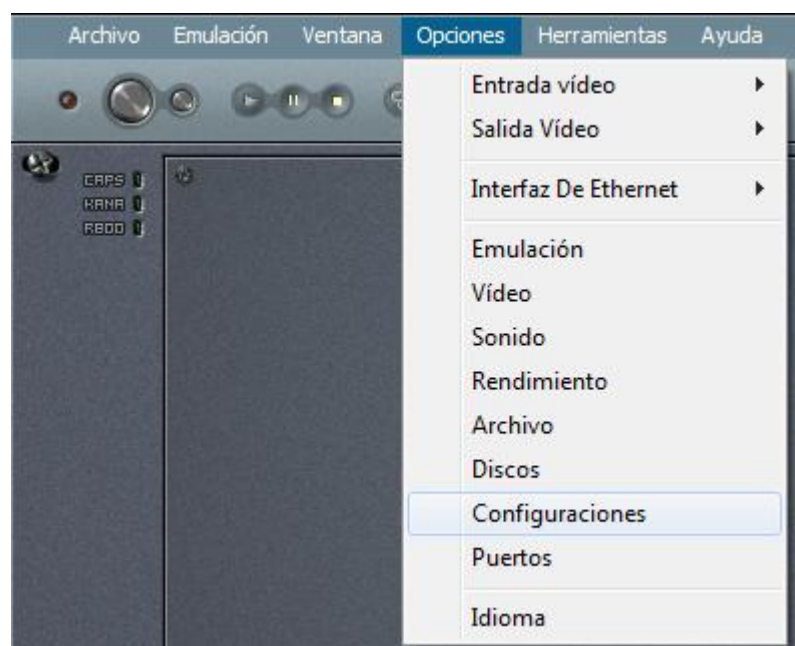
Pulsamos el Botón Next – y esperamos a que se instale el BlueMSX y en la siguiente ventana en Close.



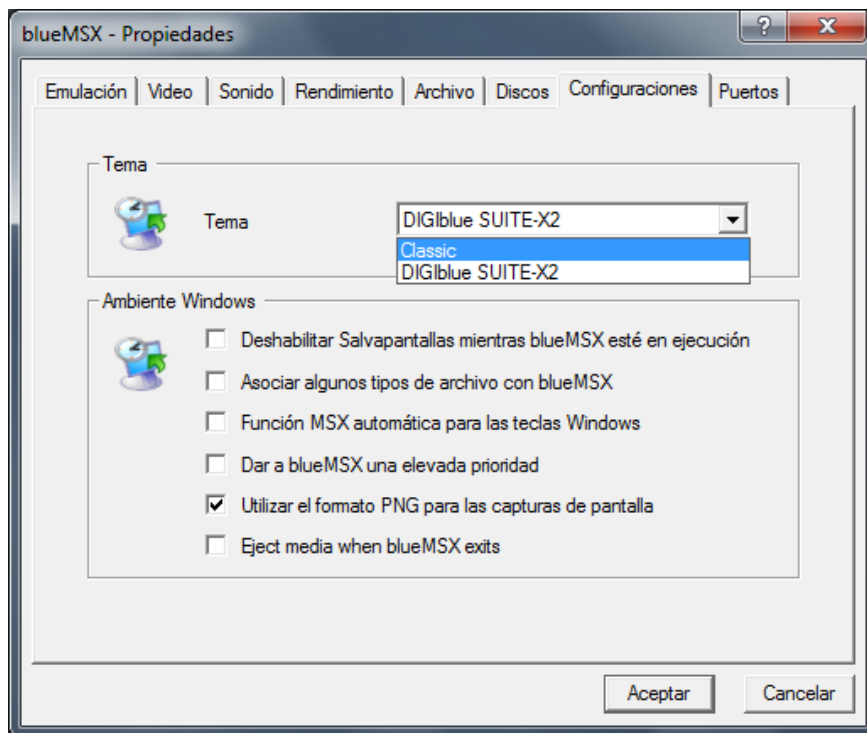
Lanzamos el programa una vez finalizada la instalación.



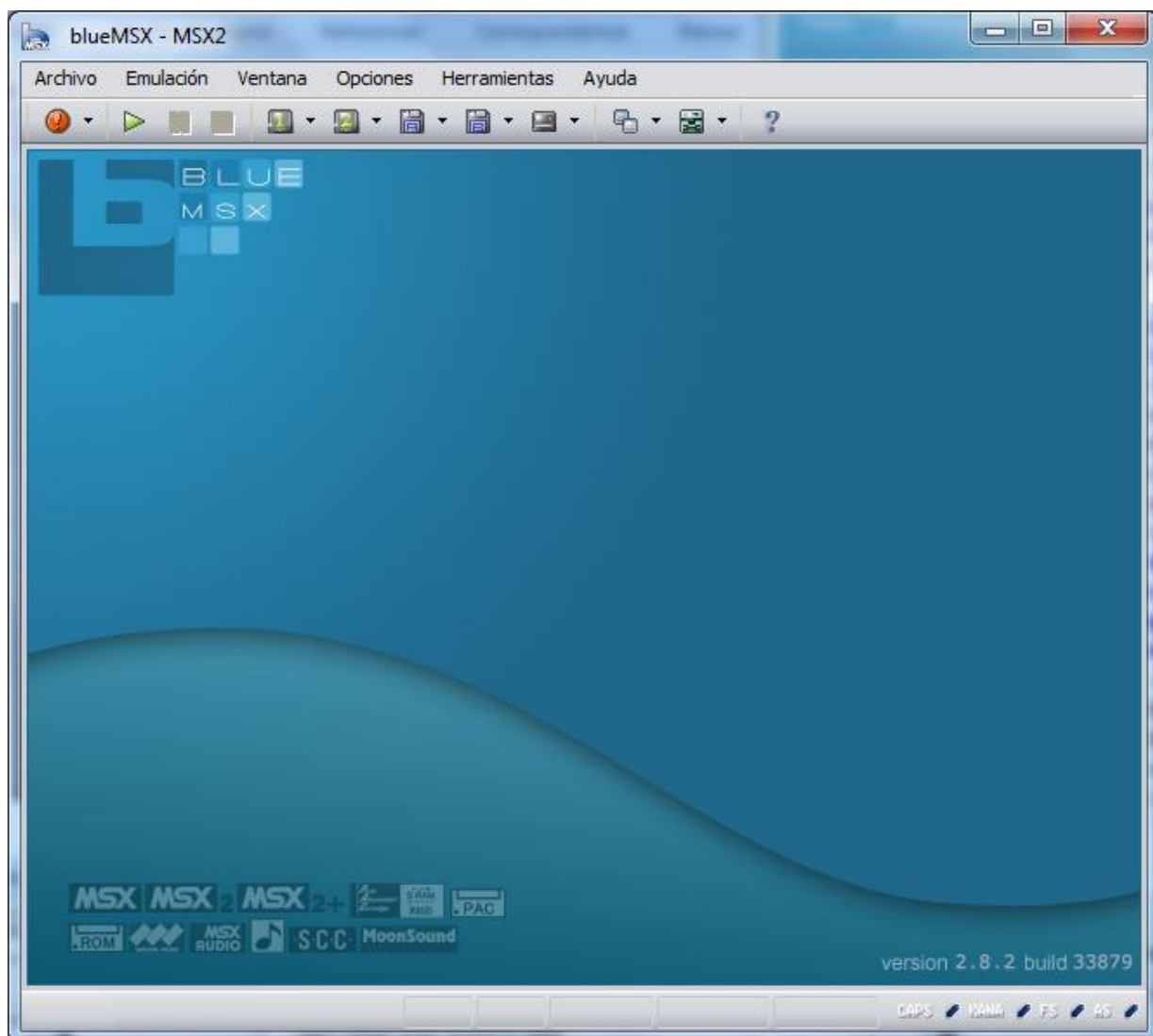
Aquí tenemos el BlueMSX con su SKIN bonito pero vamos a cambiarlo por el clásico que nos viene mejor para trabajar con el tutorial.



Pulsa en los menús de arriba del BlueMSX en Opciones y selecciona Configuraciones.

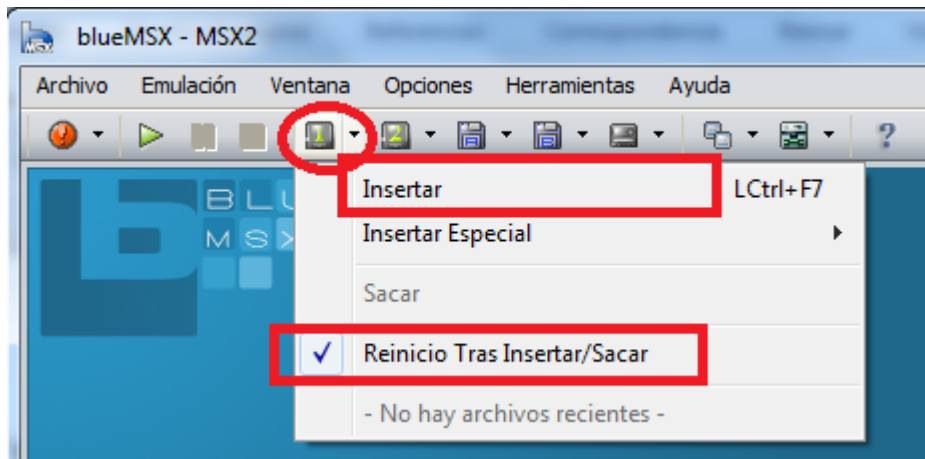


En la ventana que se abre esta que ves a la izquierda de este texto modifica el Tema pasando de DIGIblue-SUITE-X2 a Classic y pulsa el botón Aceptar.

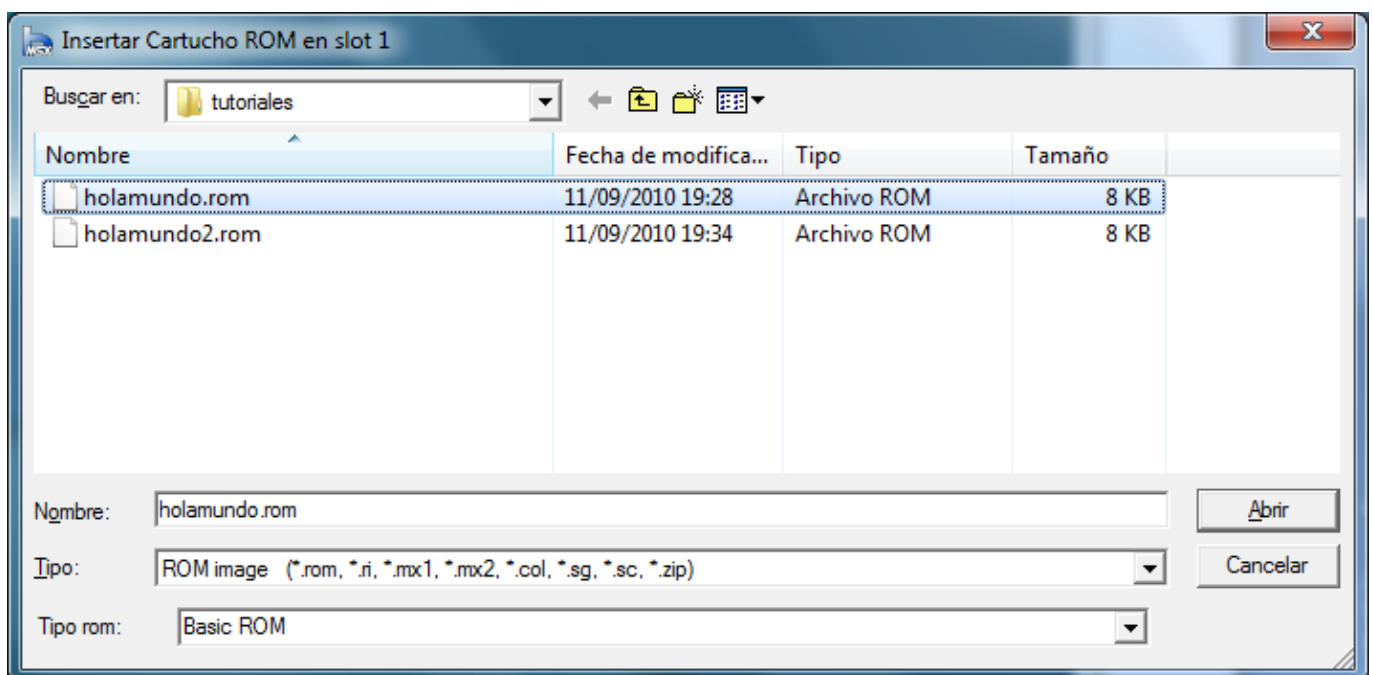


Esta va a ser la imagen que tendrá nuestro BlueMSX si has realizado bien los pasos.

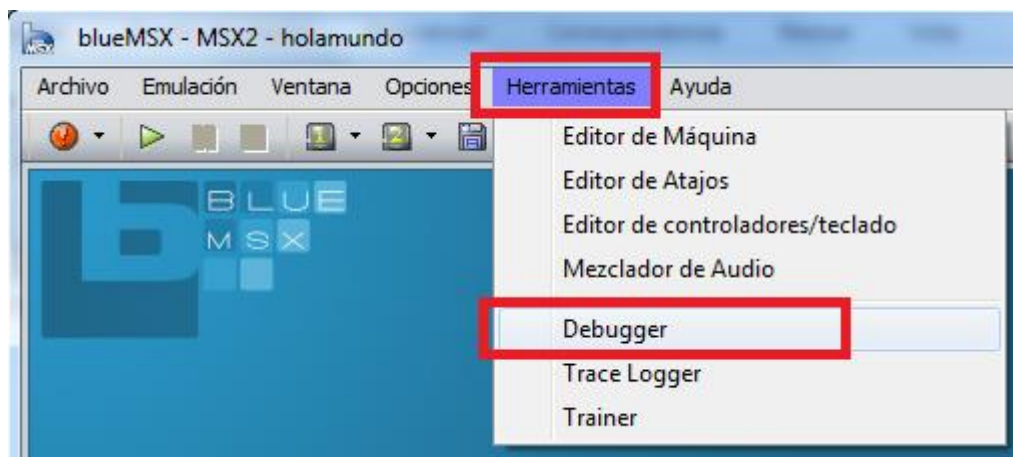
Vamos a cargar nuestra ROM “HolaMundo” en el blueMSX



Pulsa en la flechita al lado del icono del Slot de Cartuchos 1 - desmarca la V en la opción de Reinicio Tras Insertar/Sacar – vuelve otra vez y finalmente selecciona Insertar.

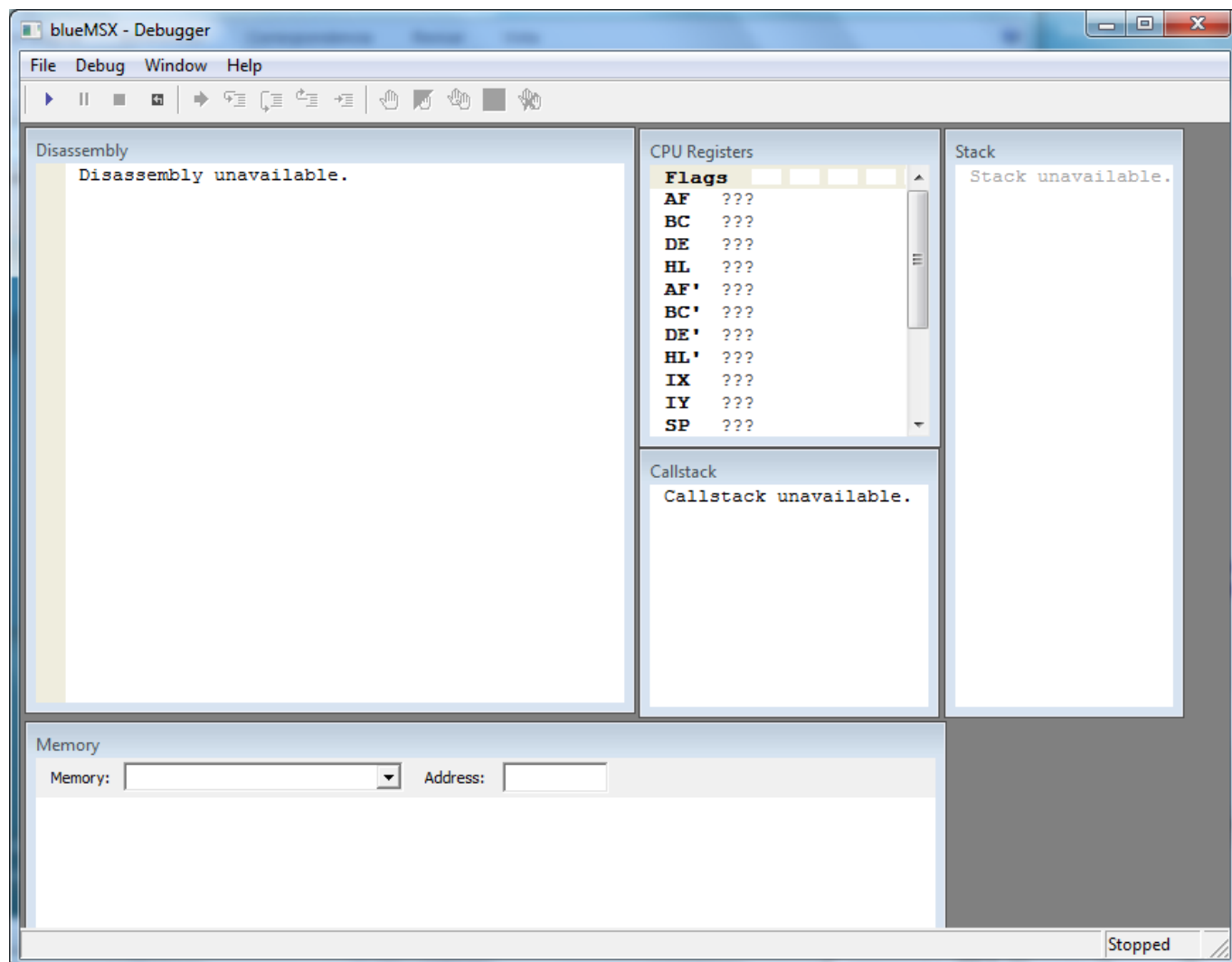


Selecciona la ROM en mi caso holamundo.rom tu cárgala desde donde la hayas guardado y con el nombre que le hayas puesto a la ROM.

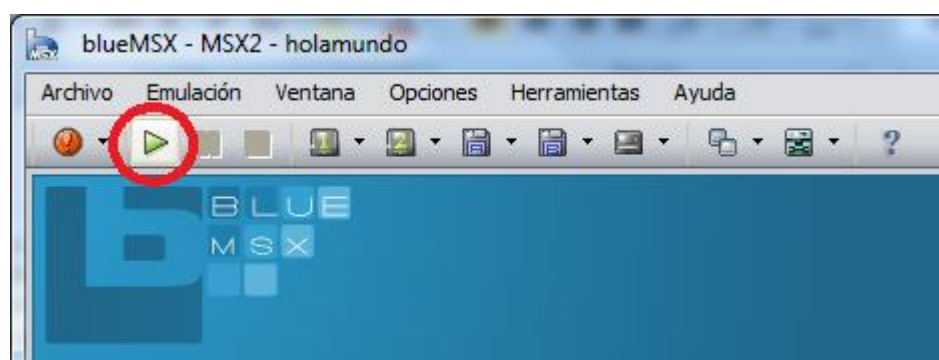


Ahora que tenemos cargada nuestra ROM Antes de ponerla en marcha vamos a abrir el DEBUGGER

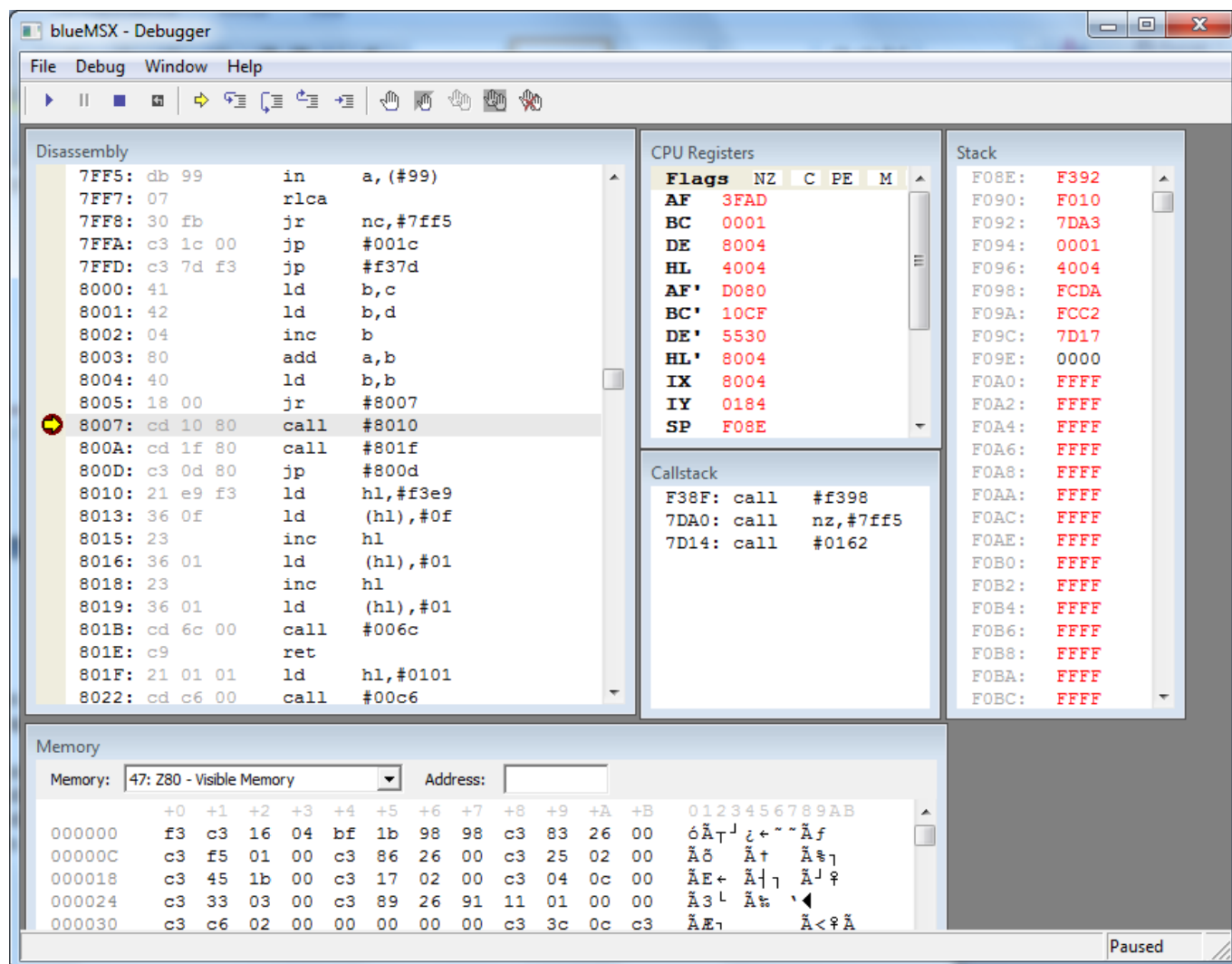
Pulsa en los menús de arriba en Herramientas y pulsa en la opción Debugger.



Aquí tenemos el Debugger lanzado pero todavía nos falta poner en marcha el BlueMSX que encenderá con la ROM o cartucho de nuestro programa y saltará al código en el punto donde hemos puesto la directiva del asMSX `.break` para que paralice la ejecución del programa en ese punto.

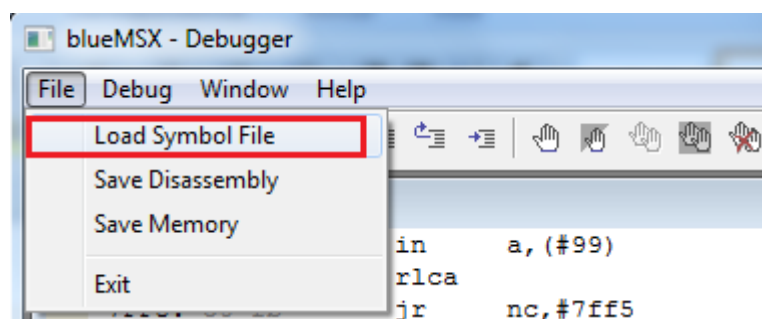


Pulsa el botón del Play en el BlueMSX y vamos a poner en marcha nuestra ROM veras que la ROM no se ejecuta y nos muestra directamente el código desensamblado en el DEBUGGER.

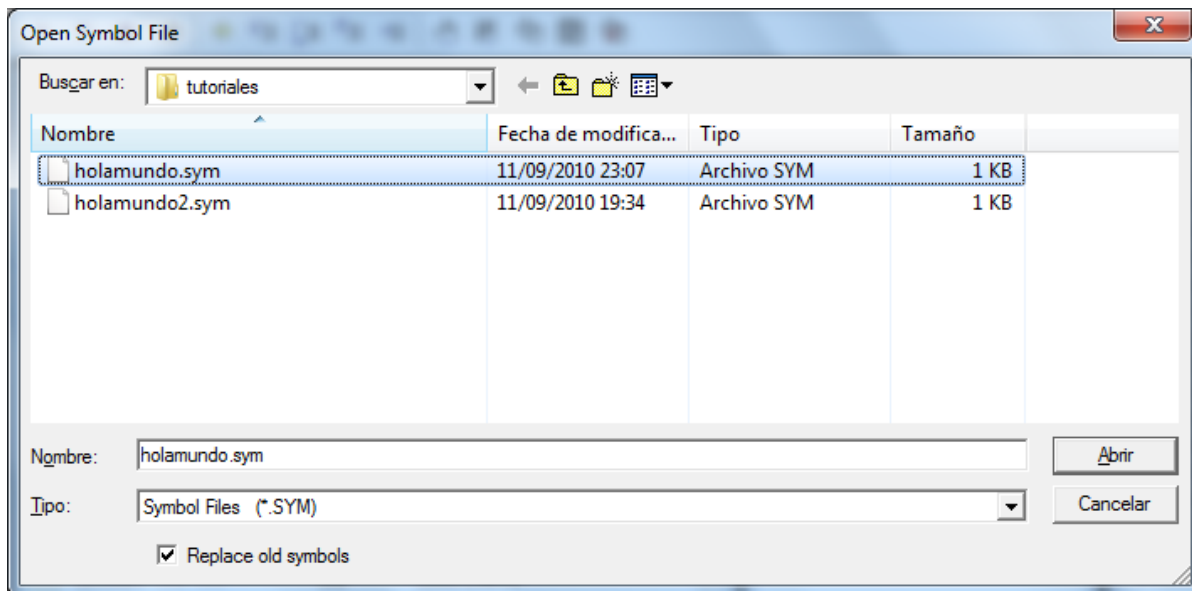


Ya tenemos nuestro programa parado en el punto que nosotros le hemos dicho. Pero qué pasa con el código no aparecen los nombres de las rutinas o de las variables. (Tranquilo esto es normal)

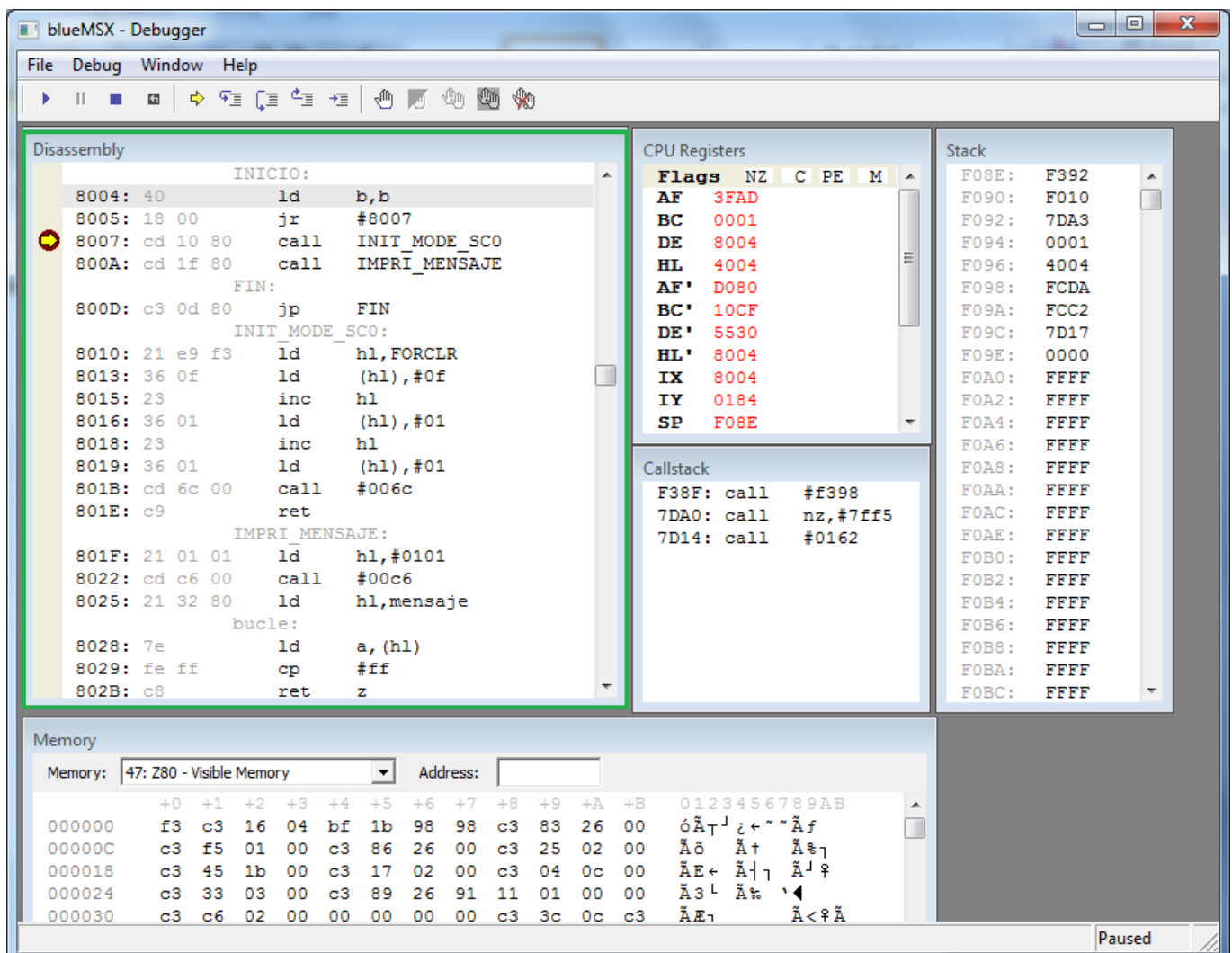
Lo que pasa es que realmente esta desensamblado el código y el no conoce los nombres ya que estos son sustituidos por direcciones de memoria a la hora de compilar, para ver el código de una manera más parecida al lenguaje ensamblador siendo nosotros los creadores del programa tenemos un fichero de símbolos creado por el asMSX a la hora de compilar, que tiene el nombre de nuestra ROM pero la terminación de este fichero es **.SYM** estará en el directorio donde hemos compilado la ROM



Pulsa en el Debugger del blueMSX en el menú **File** y selecciona la opción **Load Symbol File**



Aquí puedes ver que yo selecciono el fichero holamundo.sym que es el programa que os he explicado.

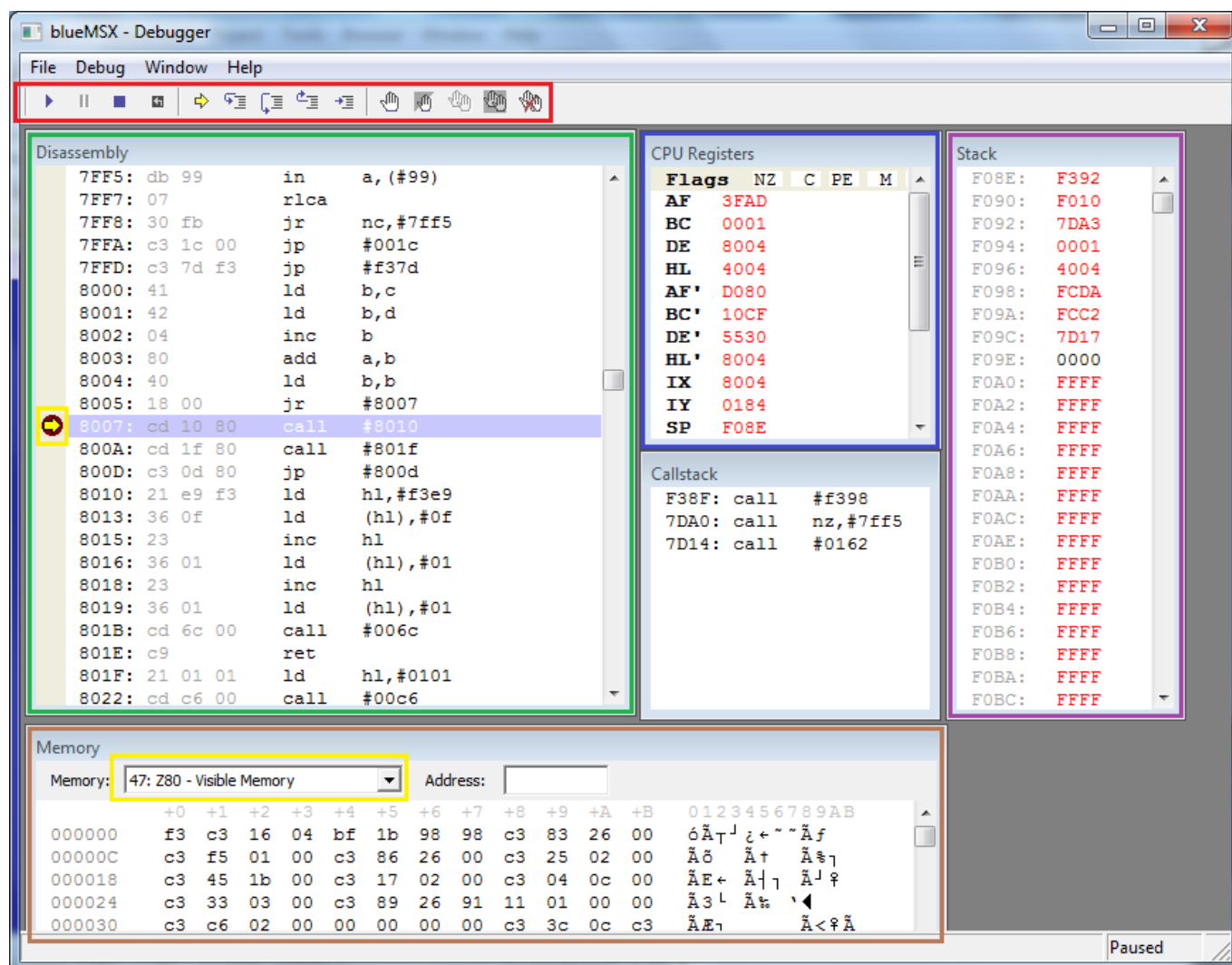


Como puedes ver en el recuadro en verde ahora ya salen los nombres de nuestras rutinas y variables si bien los valores salen en Hexadecimal, por eso yo prefiero trabajar en esta numeración ya que es la más empleada en código maquina aparte del binario claro. (Las llamadas a la BIOS no salen nombres.)

Como puedes observar el código del programa se ha detenido en el punto donde situamos el **.break** en el código, que en el desensamblado es sustituido el **.break=ld b,b jr #8007** veréis un punto rojo con una

flecha amarilla, este es el punto del programa donde nos encontramos. Los que ya estéis familiarizados con otros lenguajes de programación lo tendréis mucho más fácil de entender el apartado de seguir nuestro código paso a paso ya que todas las herramientas de desarrollo suelen incorporarlas..

Vamos a describir cada parte del Debugger del blueMSX os he puesto recuadros de colores para que sepáis de que parte de la pantalla estoy hablando, aunque me referiré a ellos aparte del color por el nombre que tienen.



1º - Color Rojo - tenemos los botones de ejecución del Debugger sobre nuestro código.

2º - Color Verde - cuadro **Disassembly** esta es la ventana donde nos muestra nuestro código desensamblado.

3º - Color Azul - cuadro **CPU Registers** esta es la ventana donde nos muestra los Registros del Z80 y algo también muy importante los Flags.

4º - Color Morado - cuadro **Stack** aquí podemos ver los valores que vamos almacenando o recuperando de la Pila.

5º - Sin color - tenemos el cuadro **Callstack** aquí puedes ir viendo las llamadas o calls que realizamos.





6º - Color Marrón - cuadro **Memory** importante este ya que vemos los valores que contiene la memoria RAM y pulsando en el cuadro Amarillo podemos cambiar y ver la memoria de video o VRAM. Etc.

7º - **Punto Rojo con flecha amarilla** - instrucción donde estamos situados en la ejecución paso a paso.






Vamos con el manejo del Debugger.








Como en los mandos de cualquier reproductor tenemos los botones que controlan el programa.

-  Botón **Play**: Pone en marcha el programa sin que se detenga salvo que tengamos un punto de interrupción fijado. (Ya explicare donde poner puntos de interrupción.)
 -  Botón **Pausa**: Deja el programa pausado sin que el código actúe.
 -  Botón **Stop**: Paraliza por completo la ejecución del código.
 -  Botón **Restart**: Carga de nuevo el programa y ejecuta de nuevo el código.
-

Apartado de ejecución del código, paso a paso o grupo de instrucciones.

-  Botón **Show Next Statement**: Cuando nos movemos en la ventana **Dissassembly** para ver más código desplazándonos arriba y abajo, este botón nos devuelve al punto donde estábamos.
 -  Botón **Step Into**: Este es el botón principal 1 ya que ejecuta el código de nuestro programa instrucción por instrucción. (Vamos lo que se conoce como paso a paso.)
 -  Botón **Step Over**: Este es el botón principal 2 ya que ejecuta los **CALL** de nuestro programa de un solo paso sin tener que ejecutar paso a paso cada instrucción, como os enseñare después.
 -  Botón **Step Out**: Ejecutara todas las instrucciones hasta que encuentre un **RET**, esto es útil cuando vamos a un **CALL** ejecutamos paso a paso y en un punto determinado todo hasta el **RET**.
 -  Botón **Run to Cursor**: El nombre lo dice todo el cursor es la franja azul que señala el set de instrucciones en **Dissassembly** ejecutara de golpe todas las instrucciones hasta llegar al cursor.
-

Apartado sobre los puntos de Interrupción de nuestro código = BreakPoint


-  Botón **Set/Remove Breakpoint**: Esto fijara o quitara un punto de interrupción en el código de nuestro programa donde este situado el cursor. (Este es el que más uso en este grupo)
 -  Botón **Enable/Disable Breakpoint**: Esto activa o desactiva un punto de interrupción en el código donde este el cursor pero no elimina el breakpoint para que sepamos donde lo pusimos.
 -  Botón **Enable All Breakpoint**: Esto activa todos aquellos puntos de interrupción que fijamos por el código pero que están desactivados. (Yo no suelo usarla al igual que la siguiente)
 -  Botón **Disable All Breakpoint**: Esto desactiva todos aquellos puntos de interrupción que fijamos por el código pero que están activados.
 -  Botón **Remove All Breakpoint**: El nombre ya lo dice pero por si acaso esto quitara todos los puntos de Interrupción de nuestro código. (Cuando digo todos son todos)
-

Esto sería todo, pero vamos a verlo con ejemplos sobre el código del primer programa.

```

      INICIO:
8004: 40          ld      b,b
8005: 18 00       jr      #8007
8007: cd 10 80   call    INIT_MODE_SC0
800A: cd 1f 80   call    IMPRI_MENSAJE
      FIN:
800D: c3 0d 80   jp      FIN

```

Aquí tenemos el principio del programa y el cursor situado en CALL INIT_MODE_SC0, pulsamos el botón  **Step Into** el programa hará una llamada a esa rutina y se parará al principio de esta.

```

      INICIO:
8004: 40          ld      b,b
8005: 18 00       jr      #8007
8007: cd 10 80   call    INIT_MODE_SC0
800A: cd 1f 80   call    IMPRI_MENSAJE
      FIN:
800D: c3 0d 80   jp      FIN
      INIT_MODE_SC0:
8010: 21 e9 f3   ld      hl, FORCLR
8013: 36 0f       ld      (hl), #0f
8015: 23          inc     hl
8016: 36 01       ld      (hl), #01
8018: 23          inc     hl
8019: 36 01       ld      (hl), #01
801B: cd 6c 00   call    #006c
801E: c9          ret

```

Stack	
F08C:	800A
F08E:	F392
F090:	F010
F092:	7DA3
F094:	0001
F096:	8004
F098:	FCDB
F09A:	FCC2
F09C:	7D17
F09E:	0000
FOA0:	FFFF
FOA2:	FFFF
FOA4:	FFFF
FOA6:	FFFF
FOA8:	FFFF
FOAA:	FFFF

Como puedes ver en las imágenes el punto de interrupción sigue al principio del programa, pero la flecha amarilla ha avanzado y se ha situado en el inicio de la rutina **INIT_MODE_SC0**. Otra cosa que debes observar es que el **Stack** ha almacenado la dirección donde tendrá que volver cuando se ejecute el **RET** que hay al final de la rutina donde estamos, esta dirección al igual que todos los cambios de valores en el Debugger los muestra en rojo **800A** que es la línea donde está el CALL IMPRI_MENSAJE.

Pulsamos de nuevo el Botón **Step Into** veras que el registro **HL** se ha cargado con el valor **F3E9** que es la dirección donde vamos a situar los colores que usaremos en la pantalla.

Recuerdas que en el código del primer programa tecleamos.

```

; Variables de sistema
      FORCLR equ    F3E9h ; Foreground colour

```

Ya que nosotros definimos que ese nombre es igual a esa posición de la memoria RAM del ordenador, en nuestro código.

CPU Registers				
Flags	NZ	C	PE	M
AF	3FAD			
BC	0001			
DE	8004			
HL	F3E9			
AF'	D080			
BC'	10CF			
DE'	5530			
HL'	8004			
IX	8004			
IY	0184			
SP	F08C			

Vamos a ver que hay inicialmente en esa posición de la memoria y veremos lo que pasa cuando seguimos ejecutando el código de nuestro programa paso a paso.

Memory															
Memory: 69: Z80 - Visible Memory															
Address: f3e9															
00F3E4	36	07	04	9f	f1	0f	04	04	c3	00	00	c3	6	•	J
00F3F0	00	00	0f	59	f9	ff	01	01	f0	fb	f0	fb	%	Y	ù
00F3FC	53	5c	26	2d	0f	25	2d	0e	16	1f	53	5c	S	\	-
-----	--	--	--	--	--	--	--	--	--	--	--	--	...		

Sitúate en la [Ventana Memory](#) del Debugger y teclea en **Address: F3E9** y pulsas Enter o Intro en el teclado, veras que te mostrara lo que hay en las posiciones de memoria que has señalado **0F 04 04** estos son los colores iniciales del Basic Blanco, Azul, Azul, si ejecutamos la siguiente instrucción paso a paso LD [HL],0Fh botón **Step Into**, veras que la posición no ha cambiado porque ya tenía ese valor, pulsamos de nuevo 4 veces **Step Into** ahora fíjate que ha cambiado a **0F 01 01** ya hemos puesto color Blanco, Fondo negro, Borde negro. 15,1,1

Ahora tenemos que situar el modo de pantalla en **SCREEN 0**. Que no es otra cosa que el **CALL #006C** en nuestro código es **CALL INITXT** que es la rutina de la BIOS que se encarga de situar el modo de la pantalla en modo texto si os fijáis en el PDF de las rutinas de la BIOS veréis que INITXT está en la posición **006Ch** **IMPORTANTE** lógicamente como ya sabemos lo que hacen las rutinas de la BIOS y estas no fallan, esta parte no hay que tracearla paso a paso. Así que usaremos el botón **Step Over** para que ejecute la rutina de una sola pasada.

Podrás observar en la ventana del BlueMSX que la pantalla ha cambiado de AZUL a NEGRO. Hemos llegado al final de la Rutina **INIT_MODE_SC0** y nos encontramos en la instrucción **RET** pulsamos el botón **Step Into** y ahora se recupera del **Stack**, a que posición de memoria tiene que regresar como vimos anteriormente a **800Ah** y al regresar a esa posición se quita del **Stack**. Situándonos en **CALL IMPRI_MENSAJE** Ahora vamos a tracear esta rutina.

```

8007: cd 10 80    call    INIT_MODE_SC0
800A: cd 1f 80    call    IMPRI_MENSAJE
      FIN:

```

Pulsamos el botón **Step Into** y nos vamos a la rutina **IMPRI_MENSAJE**

```

      IMPRI_MENSAJE:
801F: 21 01 01    ld      hl,#0101
8022: cd c6 00    call    #00c6
8025: 21 31 80    ld      hl,mensaje
      bucle:
8028: 7e          ld      a,(hl)
8029: b7          or      a
802A: c8          ret     z
802B: cd a2 00    call    #00a2
802E: 23          inc     hl
802F: 18 f7       jr      bucle
      mensaje:
8031: 48          ld      c,b

```

Como puedes ver en la imagen estamos situados al principio de esta rutina, Fíjate donde está la flecha amarilla, ahora es muy importante poner en practica lo que hemos aprendido hasta ahora con el Debugger. Fíjate en la ventana **CPU Registers** en el registro HL pulsa el botón **Step Into**. Veras que en **HL** hay **0101** ya que queremos situar texto en columna 1 Fila 1 ahora viene el CALL a la BIOS recuerdas como se ejecuta de manera completa y no paso a paso, pues pulsando el botón **Step Over** ahora vamos a recordar de nuevo como se mira la memoria para que veas donde tenemos el texto del mensaje sitúate en la ventana **Memory**.

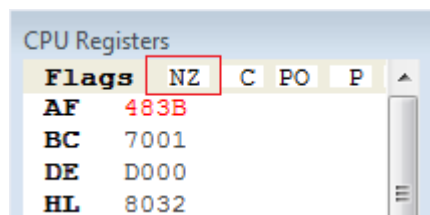
Como puedes ver arriba de este texto **mensaje** empieza en la posición **8031** de la memoria visible.

Memory													
Memory: 47: Z80 - Visible Memory		Address: 8031											
	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	0123456789AB
008028	7e	b7	c8	cd	a2	00	23	18	f7	48	6f	6c	~·ÈÍc #↑÷Hol
008034	61	20	4d	75	6e	64	6f	00	00	00	00	00	a Mundo
008040	00	00	00	00	00	00	00	00	00	00	00	00	

Pon en **Address 8031** y pulsa enter tienes que ver la imagen como la de arriba de este texto. Aquí puedes ver el texto **Hola Mundo** y los valores hexadecimales que tiene cada letra.

Si pulsamos sobre cualquier valor hexadecimal o sobre el texto podríamos modificar los valores, pero resulta que esto es memoria ROM y es de solo lectura, no podemos hacerlo te lo explico para que sepas que se pueden modificar valores, ahora eso si siempre en memoria RAM como por ejemplo cuando definamos variables en la pagina 3 dirección de memoria C000h.

Empieza el **bucle** de la rutina de impresión, esta es la encargada de pintar letra por letra y no parar hasta que se encuentre un **0**. Como tenemos el registro **HL** apuntando a la dirección donde tenemos el mensaje de texto que queremos imprimir, pulsamos **Step Into** que ejecutara el **LD A,[HL]** o lo que es lo mismo **HL=8031h** y en esa posición hay un **48h** que es la letra **H** mayúsculas, veras que el registro **AF A=48**. Ahora viene **OR A** comprobamos si **A** es cero hay que salir del bucle ejecutamos un **Step Into** y esta operación con el registro **A** activara el **Flag Z** si contiene un 0, mientras sea distinto de 0 se activara el **Flag NZ** o NoZero continuando la rutina.



Flags	NZ	C	PO	P
AF	483B			
BC	7001			
DE	D000			
HL	8032			

Esto te lo explico para futuros traceos en tu debugger.

Hay ocasiones en las que queremos tracear una rutina para saber si está funcionando bien como por ejemplo en la que estamos, donde queremos ver que todo funciona pero no esperar a que todo el bucle se ejecute, entonces tenemos la posibilidad de modificar los **FLAGS** o Banderas directamente sobre el registro del **Z80** en el caso del **OR** nos dio un **NZ** pero si pulsamos sobre el recuadro rojo del **NZ** este

cambia el **Flag** al contrario de lo que este mostrando en ese momento. Puedes realizar la prueba para que veas cómo se pueden modificar los **Flags** veras que pulsando cambia de **NZ a Z** y de **Z a NZ**.

Déjalo en **NZ** para seguir con el tutorial ahora viene la instrucción **RET Z** ejecuta **Step Into** está muy claro si el resultado del **OR A** es **Z** quiere decir que hemos llegado al final del texto y saldría de la rutina volviendo al bucle principal del programa, pero como esto no es así seguimos traceando la rutina ahora viene el **CALL** a otra rutina de la **BIOS 00A2h=CHPUT** esta rutina es la encargada de poner un carácter o letra en la posición del cursor la letra se la pasamos a la rutina en el registro **A** que ahora mismo contiene una **H** en código **ASCII la H** Mayúsculas en **48h** en Hexadecimal. Automáticamente esta rutina incrementa la posición de la columna en 1 para que la siguiente letra se pinte al lado derecho de la última que pinto. Aquí de nuevo vamos a ejecutar la rutina de la BIOS de una sola pasada sin ejecutar paso a paso, RECUERDAS COMO? Pulsa el botón **Step Over**.

Ahora se debería ver en la pantalla la primera letra del mensaje, pero todavía no entiendo porque hay veces que la pantalla cuando estamos en el debugger del BlueMSX no se actualiza bien hasta que no ejecutamos más instrucciones. (Esto es un fallo del Debugger de BlueMSX)

Seguimos con el programa ahora le toca el turno a **INC HL** ejecuta **Step Into** como ya tenemos la primera letra en pantalla incrementamos la posición de memoria para que apunte a la dirección donde tenemos la segunda letra. Veras que el registro **HL** pasa de **8031h a 8032h**

Llegamos al **JR BUCLE** con esto le decimos que el código continua ejecutándose en BUCLE. Así que pulsa **Step Into** y empezamos de nuevo de ahí que le ponga bucle ya que esto que te he explicado será lo que se irá repitiendo hasta llegar al final del texto que será cuando salga de la rutina.

Cosas que debemos aprender del debugger para futuros usos no explicados anteriormente.

- Los puntos de interrupción.

Cuando se desensambla el código de un programa o nuestro propio código podemos poner puntos de interrupción en cualquier parte de la memoria, esto se suele realizar para varios cometidos, pero te explico uno. Imagina que quieres ver como funciona una determinada rutina, pero quieres solo tracear esa rutina y no todas las rutinas que hay anteriormente, así pues ponemos un punto de interrupción en esa rutina y ejecutamos el programa, este ira ejecutando de manera automática todo hasta detenerse en el punto de interrupción que hemos fijado. Veámoslo con un ejemplo.



Pulsa el botón **Restart** para que se cargue y se ejecute de nuevo el código de nuestro programa.

Si has cerrado todo ejecuta de nuevo el blueMSX lanza de nuevo el debugger, y carga el programa.


```

      INICIO:
8004: 40          ld      b,b
8005: 18 00       jr      #8007
8007: cd 10 80    call    INIT_MODE_SC0
800A: cd 1f 80    call    IMPRI_MENSAJE

      FIN:
800D: c3 0d 80    jp      FIN

```

Como en nuestro código pusimos un punto de interrupción con la orden **.BREAK** este se para siempre en este punto, vamos a quitar este punto de interrupción pulsando una vez con el ratón sobre el punto rojo con la flecha amarilla.

```

      INICIO:
8004: 40          ld      b,b
8005: 18 00       jr      #8007
8007: cd 10 80    call    INIT_MODE_SC0
800A: cd 1f 80    call    IMPRI_MENSAJE

      FIN:
800D: c3 0d 80    jp      FIN

```

Como puedes ver en la imagen ya hemos quitado el punto de interrupción inicial, de igual manera que quitamos el punto de interrupción podemos ponerlo, siempre que pulsemos con el ratón en esa parte.

```

      INICIO:
8004: 40          ld      b,b
8005: 18 00       jr      #8007
8007: cd 10 80    call    INIT_MODE_SC0
800A: cd 1f 80    call    IMPRI_MENSAJE

      FIN:
800D: c3 0d 80    jp      FIN

      INIT_MODE_SC0:
8010: 21 e9 f3    ld      hl, FORCLR
8013: 36 0f       ld      (hl), #0f
8015: 23           inc      hl
8016: 36 01       ld      (hl), #01
8018: 23           inc      hl
8019: 36 01       ld      (hl), #01
801B: cd 6c 00    call    #006c
801E: c9          ret

      IMPRI_MENSAJE:
801F: 21 01 01    ld      hl, #0101
8022: cd c6 00    call    #00c6
8025: 21 32 80    ld      hl, mensaje

```

Ahora Fijamos el punto de interrupción justo donde empieza la rutina de impresión del mensaje, como puedes ver en la imagen encima de este texto.

Ahora pulsamos en el botón de **Start / Continue**  para que se ejecute el código de nuevo.

```

      IMPRI_MENSAJE:
801F: 21 01 01    ld      hl, #0101
8022: cd c6 00    call    #00c6
8025: 21 31 80    ld      hl, mensaje

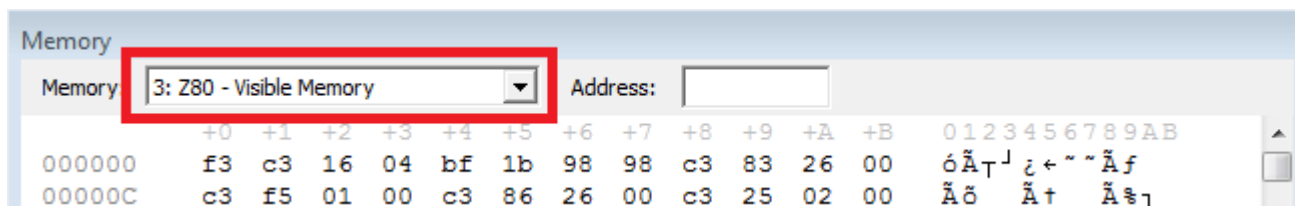
      bucle:
8028: 7e          ld      a, (hl)
8029: b7          or      a
802A: c8          ret      z
802B: cd a2 00    call    #00a2
802E: 23          inc      hl
802F: 18 f7       jr      bucle

```

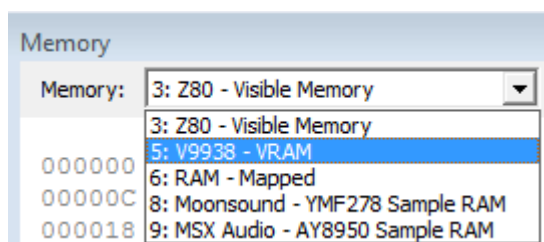
Como puedes ver se ha ejecutado todo el código anterior hasta detenerse donde hemos fijado el punto de interrupción.

Con esto damos por finalizado la sección sobre puntos de interrupción.

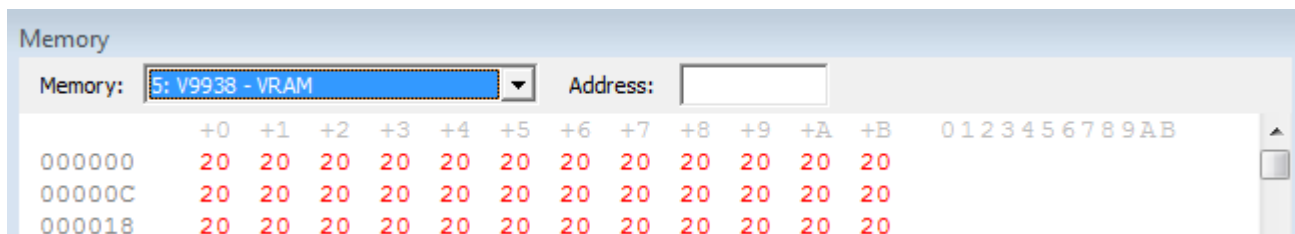
- Manipulación y visualización de datos en la memoria.



Ya hemos visto anteriormente como ver los valores hexadecimales en la memoria y como modificar estos valores, pero quería explicaros cuando empecemos a trabajar con la memoria de video o sprites que para poder ver si nuestras rutinas están trabajando bien sobre la VRAM también podemos ver los valores que estamos escribiendo en la memoria de video, para este cometido tendremos que cambiar en el recuadro que te remarco en rojo, de la ventana [Memory](#) cambiamos de la memoria RAM del Z80 a la Memoria de video VRAM.



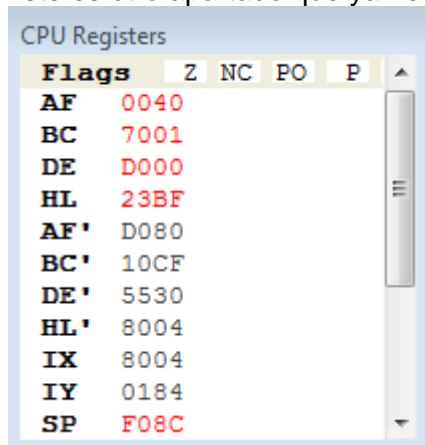
Pulsando en el flecha puedes ver la [RAM z80 – Visible Mem](#). Selecciona [v9938 – VRAM](#) para ver los valores de la memoria de video. También se podría ver la memoria mapeada o la los datos del chip de sonido.



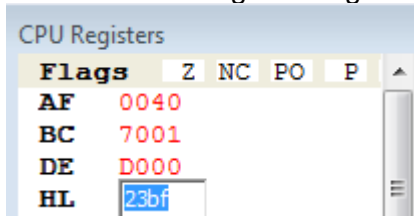
Como puedes ver en la imagen ya estaríamos viendo los valores hexadecimales de la [VRAM](#).

- Manipulación de FLAGS (Banderas) o Registros del Z80

Este es otro apartado que ya hemos visto pero que voy a puntualizar un poco más.



Al igual que has aprendido a modificar los Flags a nuestro antojo en determinadas ocasiones de nuestro traceo nos puede interesar modificar los valores de los registros, esto lo puedes realizar pulsando con el ratón sobre el valor que contiene el registro que queremos modificar en esta imagen el registro HL y poner el valor.

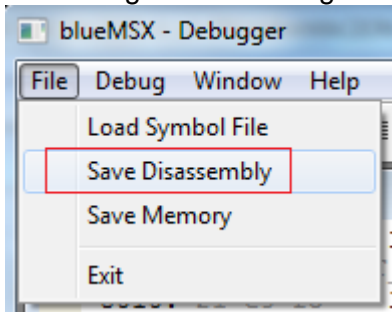


- Volcado de la memoria y desensamblado

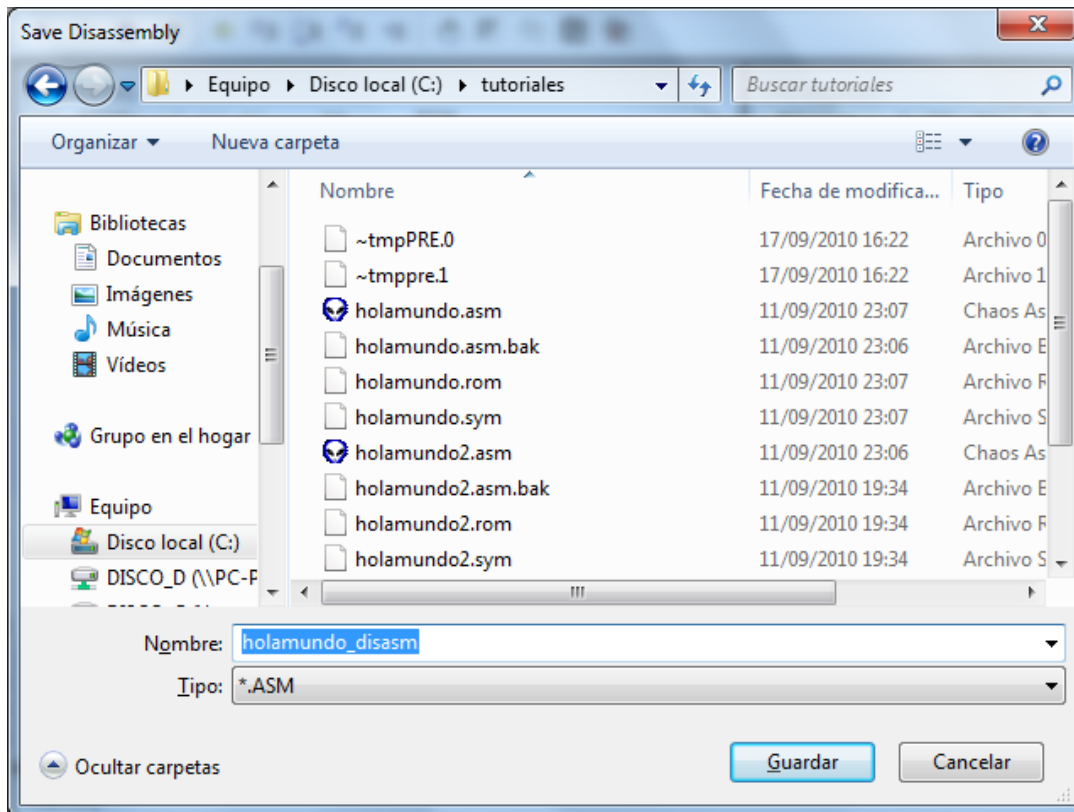
Yo siempre he dicho que se puede aprender mucho de cómo se realizan los programas viendo el código realizado por otros programadores profesionales o normales.

Una parte es esta el BlueMSX incorpora una opción que es desensamblar y guardar el código en un fichero de texto. Y la otra os la explico después.

Vamos a guardar el código desensamblado de nuestro programa.

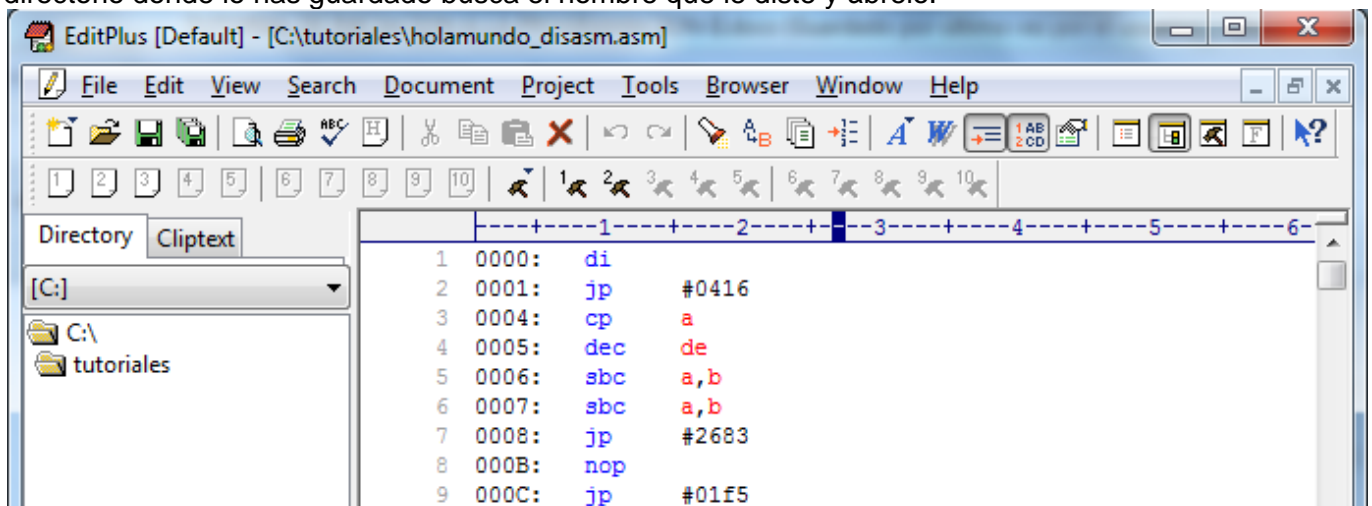


Pulsa en el menú File en el Debugger del blueMSX
Y selecciona la opción Save Disassembly

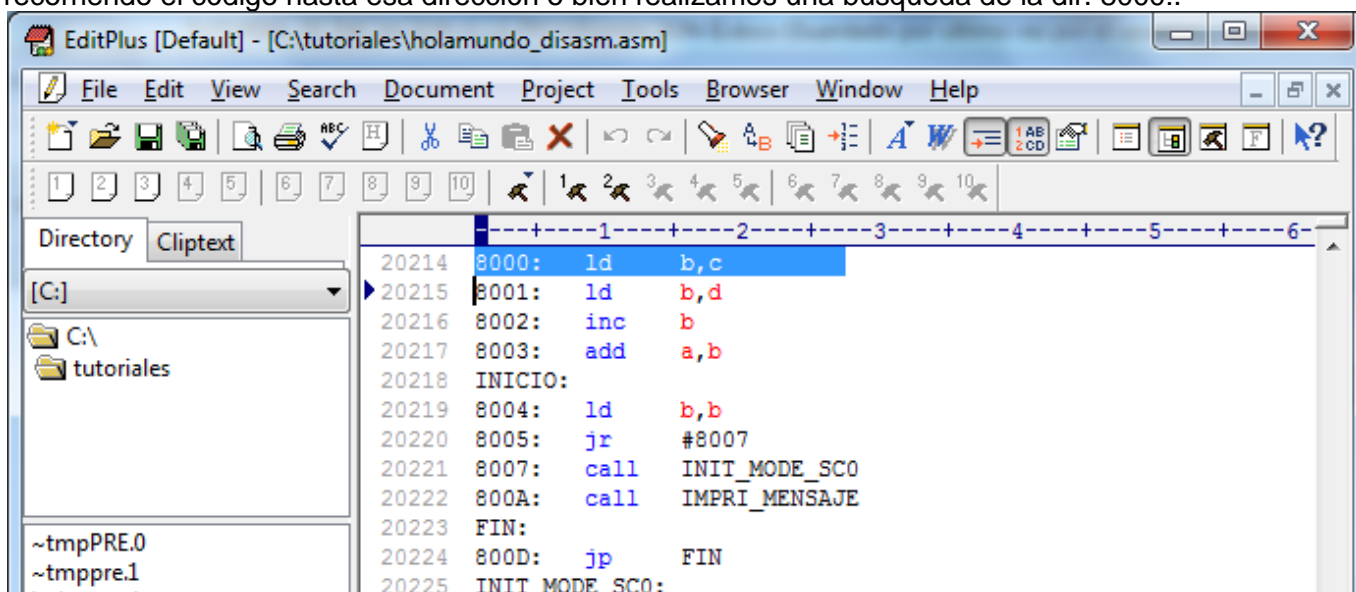


Dale un nombre al programa que queremos guardar el desensamblado y pulsa el botón Guardar.
Deberían ponerle una opción al blueMSX para decirle desde y hasta que dirección de memoria quieres desensamblar ya que el blue lo hace desde la dirección de memoria 00000 hasta la 65535 todo vamos.

Ahora abre el programa EditPlus y carga el fichero guardado con el debugger del blueMSX, vete al directorio donde lo has guardado busca el nombre que le diste y ábrelo.



Aquí en la imagen de arriba puedes ver que el código desensamblado empieza en la dirección 0000 como nosotros sabemos que nuestro código empieza en la dirección 8000h Hexadecimal nos vamos recorriendo el código hasta esa dirección o bien realizamos una búsqueda de la dir. 8000:.

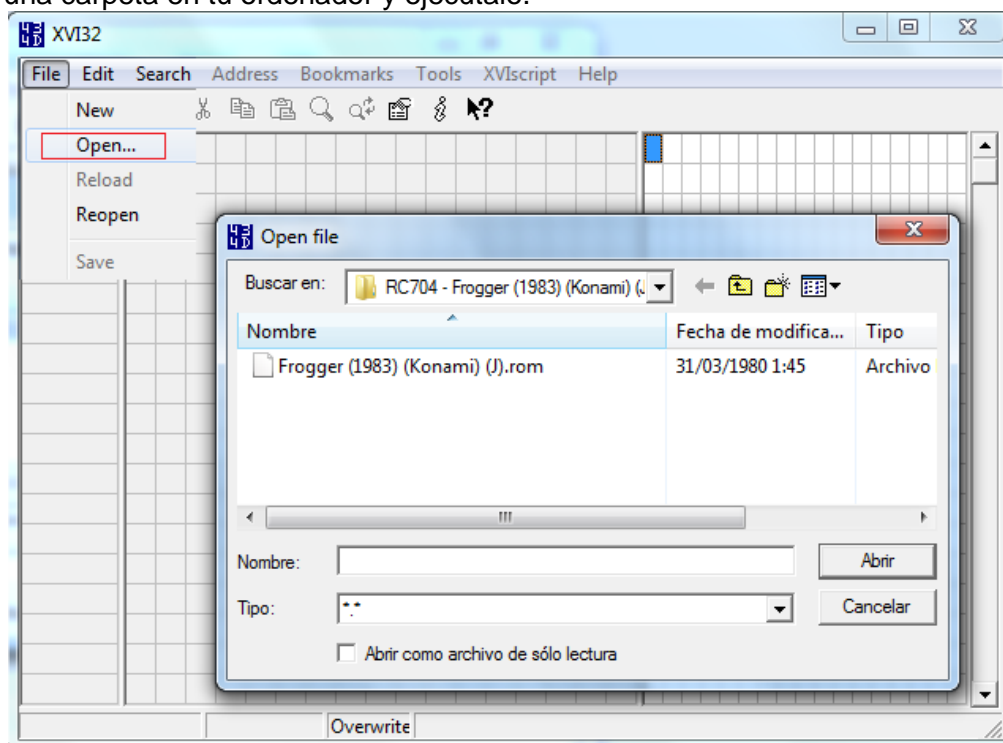


Siendo un documento de texto puedes borrar las partes que no te interesan y quedarte con lo que te interesa del desensamblado, ahora eso si aquí nos ha puesto las etiquetas porque teníamos cargado el fichero SYM en el debugger en un desensamblado normal solo saldrían posiciones de memoria.

Ahora veamos como cargar y trazar una ROM comercial de KONAMI.

Este es el objetivo [Frogger \(1983\) \(Konami\) \(J\).rom](#) es una ROM pequeña de 8KB pero inmensa en código, como es una rom y no tenemos el código fuente en ensamblador pues no conocemos ciertos datos que nos hacen falta así como poner un .break en el código para parar la ejecución del programa, vamos por pasos lo primero es saber en qué posición de memoria empieza el código del [Frogger](#) lo averiguamos de la siguiente manera.

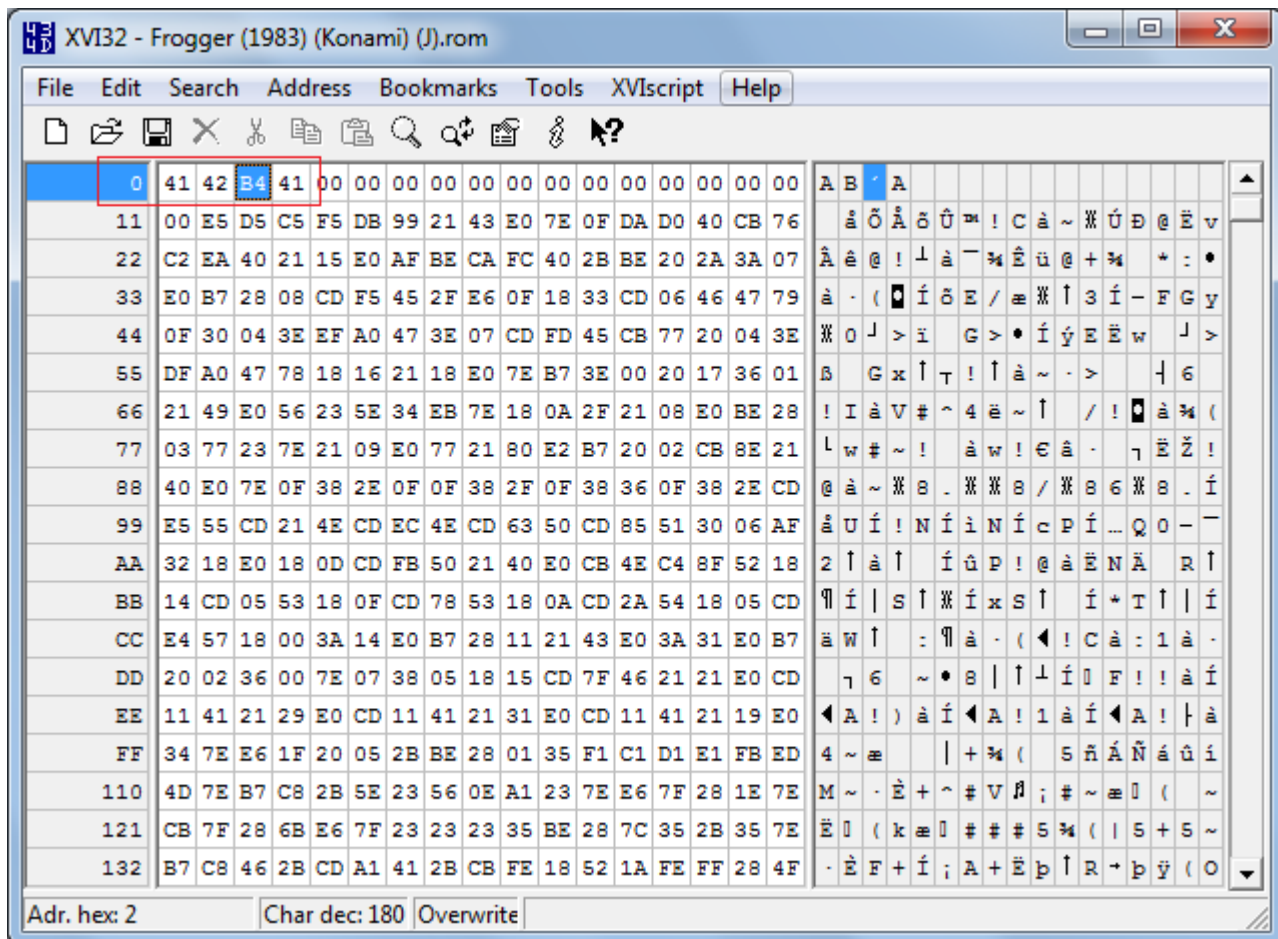
1 – Si no tienes ningún editor hexadecimal en tu ordenador puedes descargar este programa gratuito desde este enlace <http://www.handshake.de/user/chmaas/delphi/download/xvi32.zip> descomprímelo en una carpeta en tu ordenador y ejecútalo.




Vete al [menú File](#) y selecciona [Open...](#)

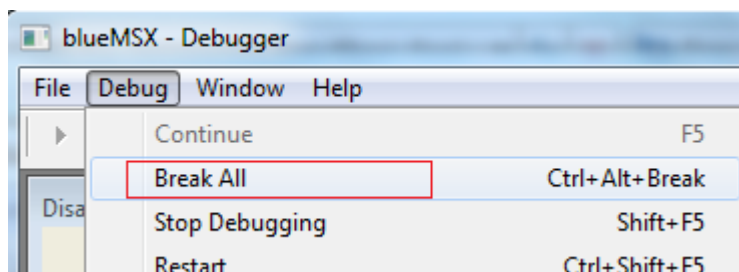
En el cuadro de dialogo [Open file](#) busca donde tengas bajada la ROM del Frogger selecciónala y dale al [botón Abrir](#).

No preguntéis en el foro-blog donde se localizan las ROM

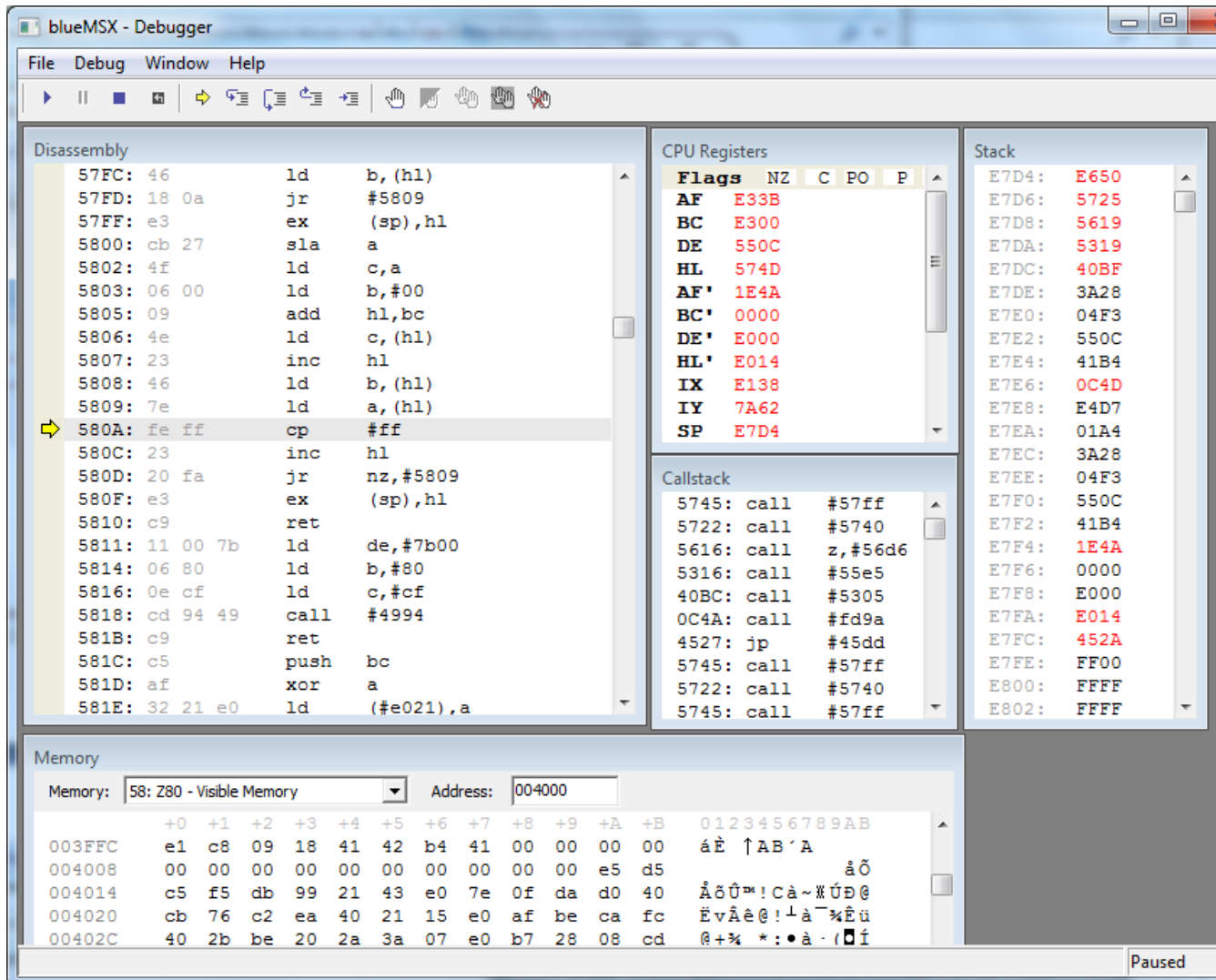


Este es el aspecto que tiene que tener el programa una vez que hemos abierto la ROM del **Frogger** de **Konami**. Aquí en el recuadro en Rojo esta el KIT de la cuestión todas las ROMs empiezan por los valores **41 42** seguido de la dirección donde empieza el código del programa de esta manera sabemos que el código empieza en la dirección **41B4h** (y dirás de donde saco eso) las direcciones de memoria siempre se almacenan así, el **byte bajo** primero y el **byte alto** después de esta manera la dirección 8000h se almacena como 00 80 así pues como después del 41 42 hay un B4 41 esto quiere decir que el código empieza en la dirección 41B4h (Espero que esto lo tengáis claro).

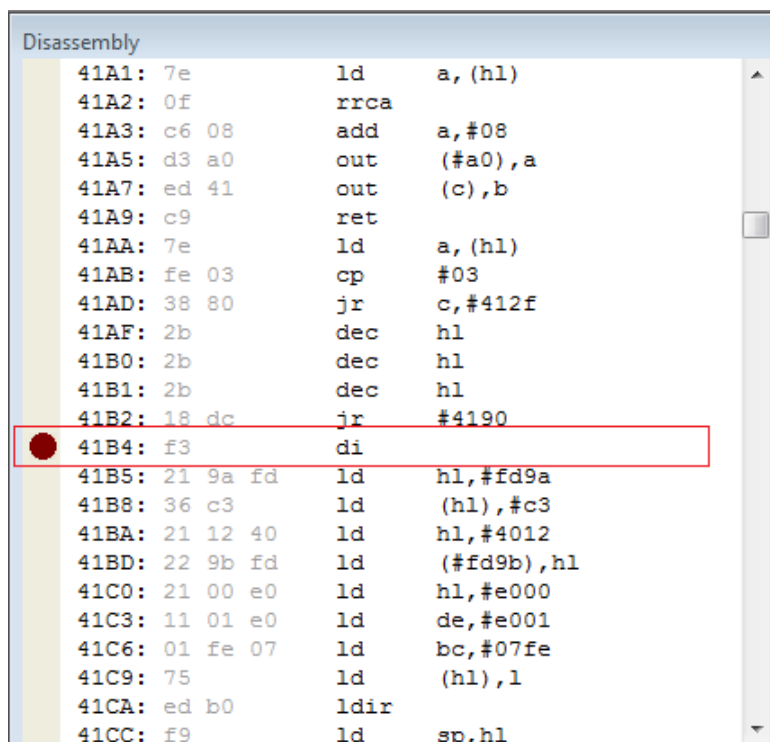
Ya podéis cerrar este programa ya que hemos averiguado nuestro primer objetivo saber el inicio del código. Ahora vamos a empezar a tracear el juego. Abre el blueMSX y carga la ROM del Frogger, después abre el debugger y dale al botón del Play  en el blueMSX para que se ejecute la ROM, veras que esta vez no ha entrado directamente al Debugger porque no tenemos ningún punto de interrupción (y si no podemos poner en el código un .break como hacemos para que el programa se detenga.... Tranquilos no pasa nada que esta todo previsto.)




Estando en el Debugger pulsa en el **menú Debug** y selecciona la opción **Break All** esto hará que se produzca una interrupción donde se encuentre en ese momento el código de ejecución deteniéndose en ese punto.



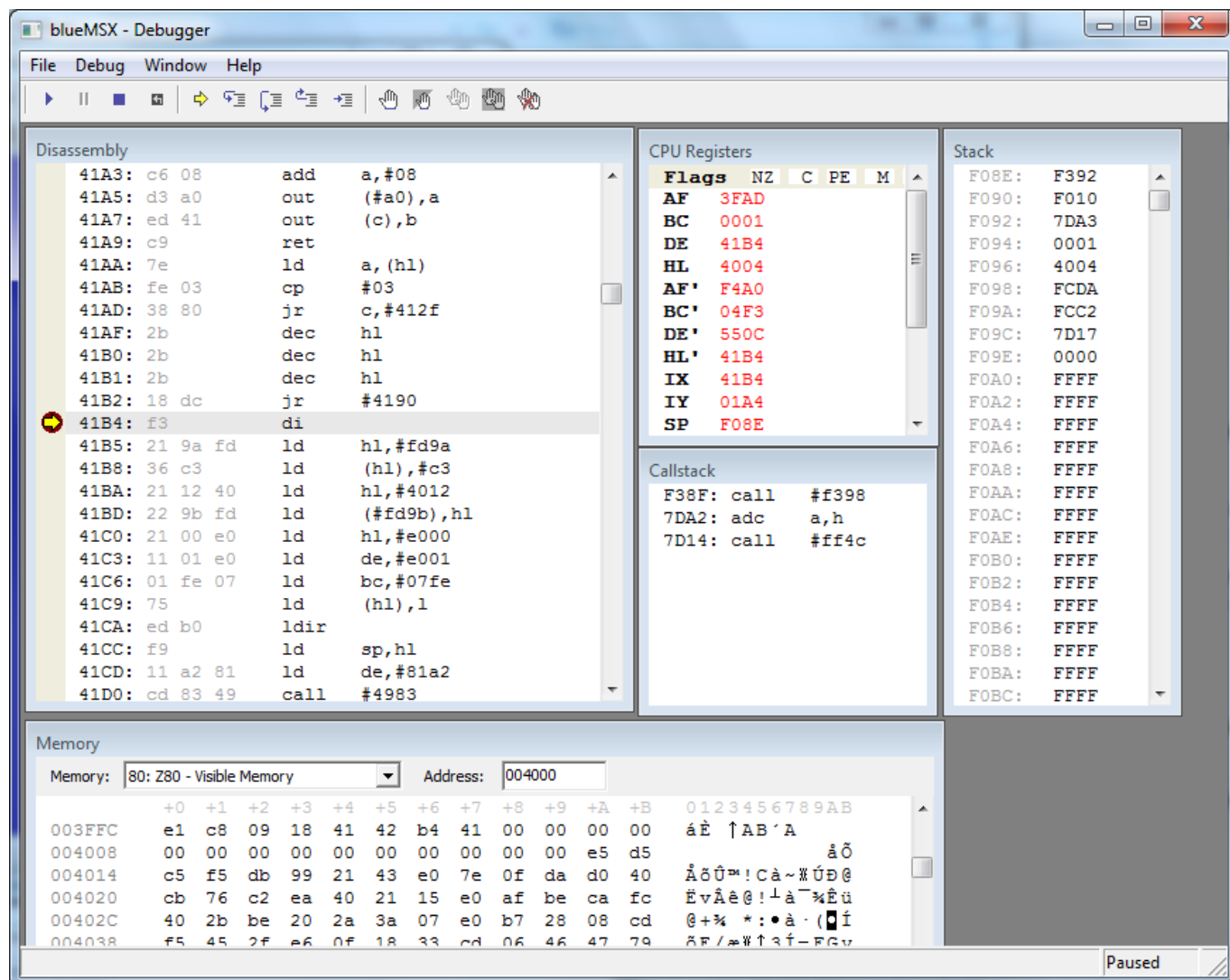
Ya tenemos el código detenido, ahora nos vamos a la posición de memoria **41B4H** para colocar un punto de interrupción en esa dirección que es donde empieza el código juego Frogger.



Desplázate por la ventana Disassembly hasta que localices la dirección de memoria 41B4h y coloca un punto de interrupción en esa posición de memoria.

Ahora para empezar el traceo desde 0 pulsa en el botón  Restart. Esto hará que el juego se ejecute de nuevo desde el principio.

Pero al haber puesto el punto de interrupción en el principio del código hará que el debugger se detenga en este punto.



Aquí tienes el [Frogger](#) listo para empezar a tracear. Practica todos lo que quieras sobre lo que has aprendido en estas lecciones sobre el debugger poniéndolo en práctica con esta ROM. Pero llegar a entender todo el código de un programa es una tarea muy compleja.

Bueno ya no te explico más cosas sobre el debugger porque si no va a ser un manual sobre este programa en vez de un tutorial sobre programación en ensamblador para el MSX. Con esto tienes todo lo básico que te hace falta para tracear programas y ver los errores.

Espero que haya sido de vuestro total agrado y nos vemos en el próximo tutorial. Donde explicare como realizar gráficos e incorporarlos al código fuente de nuestro programa. Así como los programas que utilizo para este cometido.

José Vila Cuadrillero

"ES DETESTABLE ESA AVARICIA ESPIRITUAL QUE TIENEN, LOS QUE SABIENDO ALGO, NO PROCURAN LA TRANSMISION DE ESOS CONOCIMIENTOS."

Miguel de Unamuno

Escritor y Filósofo.

(Bilbao 1864 - Salamanca 1936)

TUTORIAL DE INICIACION A LA PROGRAMACION EN LENGUAJE ENSAMBLADOR PARA MSX

3ª PARTE – EL MUNDO DE LOS GRAFICOS 1

En esta parte del tutorial vamos a crear un [Hola Mundo Grafico](#) utilizando un juego de caracteres de texto creado por nosotros y todo lo relacionado con el modo grafico Screen 2, así como las herramientas que necesitaremos para realizar todo este cometido.

Antes de empezar con el código del [Hola Mundo Grafico](#), vamos con la parte que menos le gusta a la gente, la teoría... pero ojo que sin ella no se comprenden las cosas.

A diferencia del primer ejemplo donde solo seleccionábamos la fila columna y el texto que queríamos imprimir, y este salía impreso en la pantalla.

Recuerda que el modo de pantalla seleccionado era [Screen 0](#) con el siguiente código.

```
call INITXT ; BIOS set SCREEN 0
```

Este es el modo de pantalla que utiliza el [Basic del MSX](#), el set de caracteres o letras que sale impreso en pantalla está integrado en la BIOS, y es el que se muestra cuando encendemos el ordenador.

Ahora que vamos a utilizar el [Screen 2](#) este es un modo grafico, y podremos crear las letras como queramos dándoles un colorido especial, haciendo que nuestra ROM tenga un aspecto más profesional. (Tengo que aclarar que todo lo aquí explicado no solo sirve para crear letras, también sirve para crear los gráficos que utilizaremos en nuestras ROM's, ya que el procedimiento es el mismo.)

MODOS EN MSX1

SCREEN 0: texto de 40 x 24 con 2 colores

SCREEN 1: texto de 32 x 24 con 16 colores

SCREEN 2: gráficos de 256 x 192 pixeles con 16 colores

SCREEN 3: gráficos de 64 x 48 pixeles con 16 colores



Aquí tienes muestras de diferentes sets de caracteres de varios juegos.

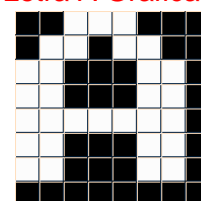
Letra A en Binario

00111000 - Byte 0
01101100 - Byte 1
11000110 - Byte 2
11000110 - Byte 3
11111110 - Byte 4
11000110 - Byte 5
11000110 - Byte 6
00000000 - Byte 7

La pantalla en [Screen 2](#) está compuesta de [256 Pixeles en horizontal](#) por [192 pixeles en vertical](#) pero si lo miramos en caracteres de [8x8](#) pixeles hablamos de [32](#) caracteres en horizontal por [24](#) en vertical.

Si multiplicamos [32x24x8](#) bytes que tiene cada carácter nos da un total de [6144 Bytes - 1800h](#) son [6Kb](#). Estos datos gráficos se almacenan en la Video Ram o VRAM desde la posición [0000h](#) hasta la posición [17FFh](#) en el mundo del MSX se le llama a esta zona [Character Pattern Table](#).

Letra A Grafica

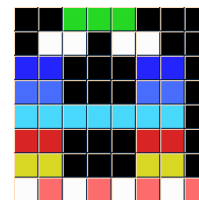


Para que lo entiendas mejor te pongo una imagen que vale más que mil palabras.

Esto es un carácter de [8x8 Pixeles](#) son 8 bytes ya que un byte son [8 bits](#) el ancho por 8 bytes de alto. Por eso multiplico los 32 caracteres de ancho por los 24 caracteres de alto por los 8 Bytes que tiene cada carácter. En total 6144 Bytes 1800h en Hexa.

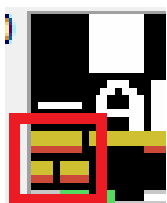
(Puedes ver el carácter A en modo grafico y en modo binario [00111000 – Byte 0](#))

El color en los caracteres según el estándar del **MSX1** se tiene que ajustar a un máximo de **2 colores por cada byte**, un color para el fondo y otro para la tinta de un total de **16 colores disponibles**. Nuevamente te pongo la misma imagen del carácter pero con una prueba de color, en el primer byte el fondo es negro y la tinta verde. El segundo byte el fondo es negro y la tinta blanca, pero si te fijas en el último byte del carácter el fondo es blanco y la tinta rosa.



Estos datos gráficos se almacenan en la VRAM desde la posición **2000h** hasta la posición **37FFh** en el mundo del MSX se le llama a esta zona **Colour Table** y ocupa lo mismo que la **Character Pattern Table** ósea $32 \times 24 \times 8 = 6144$ Bytes - 1800h en Hexadecimal (**Siempre procuro trabajar con esta nomenclatura**)

Aclarado este tema que es bastante importante ya que en el MSX todo el tema grafico funciona con **caracteres**, abreviado **CHR** del Ingles **CHaRacter**. Míralo de esta manera si cada pantalla grafica ocupa 6 Kb. sin color y tenemos una ROM de 32 Kb pues nos da para 5 pantallas sin dejarnos espacio para el color, los sprites, la música, y la programación etc. Por eso todas las ROM's o los juegos utilizan un juego de CHRs o Tiles (Azulejos) y con ellos se construye un mosaico a base de repetir Tiles por las diferentes zonas de las pantallas que iremos creando. (**Vamos con las imágenes para que lo entiendas**)



Como puedes ver en la pantalla del **King's Valley** de Konami tenemos un carácter el de los ladrillos amarillos que se repite por diferentes zonas de la pantalla al igual que el de las escaleras, a esta técnica de utilizar caracteres repetidos para crear pantallas se le llama mapeado. La pantalla de la derecha es la **Character Pattern Table** junto con la **Colour Table** que son los gráficos que los dibujantes de Konami crearon, mientras que el mapeado o Tileado en MSX se conoce como **Name Table**. La imagen de la izquierda.

Estos datos se almacenan en la VRAM desde la posición **1800h** hasta la posición **1AFFh** esta zona ocupa 32×24 caracteres = 768 Bytes – 300h Hexa. Con esta técnica podemos crear muchas más pantallas y si comprimimos los datos de los gráficos y el mapeado pues mejor.

Ahora la parte más difícil de explicar. Si te fijas de nuevo en la pantalla de la derecha veras que los CHRs o Tiles están repetidos 3 veces en la **Character pattern Table** y en la **Colour Table** esto es debido a una limitación del MSX, sabemos que la pantalla está compuesta de 32 CHRs de ancho por 24 CHRs de alto en total 768 caracteres, **pero lo que no sabemos es que a su vez está dividida en 3 tercios** o bancos de 256 CHRs. Fíjate en la imagen de la derecha en su izquierda veras **p0, p1, p2** estos son los 3 bancos de caracteres cada banco está compuesto de 32 CHRs de ancho x 8 CHRs de alto = 256 CHRs.

Pregunta: Entonces no puedo utilizar más de 256 CHRs porque tengo que copiarlos en los 3 bancos?

Respuesta: NO. Si un CHR se ha de emplear en toda la pantalla, si lo has de tener en los 3 bancos pero si un CHR solo lo usas en la zona del primer banco, pues solo lo tienes que colocar en ese banco, hay que ir colocando los CHRs en los bancos según la zona de la pantalla donde los tengamos que utilizar, además a medida que cambiamos de niveles podemos ir incorporando nuevos banco de CHRs o ir agregando nuevos CHRs a los bancos que ya tenemos en VRAM a medida que los necesitemos.

Vamos con las imágenes que siempre nos revelan o nos aclaran más las cosas que con texto.

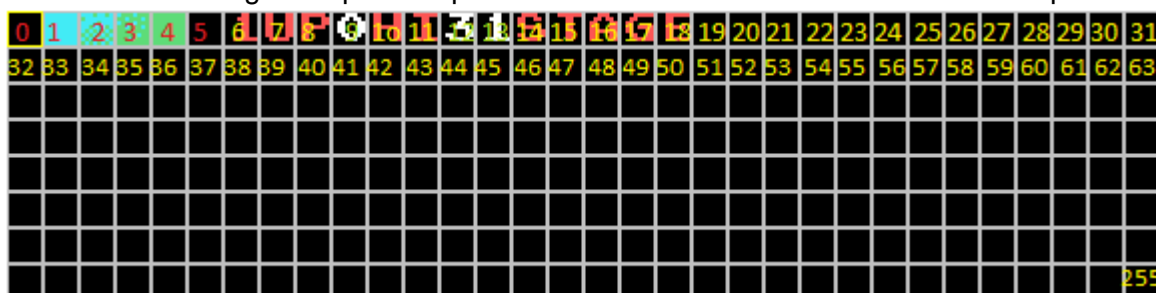
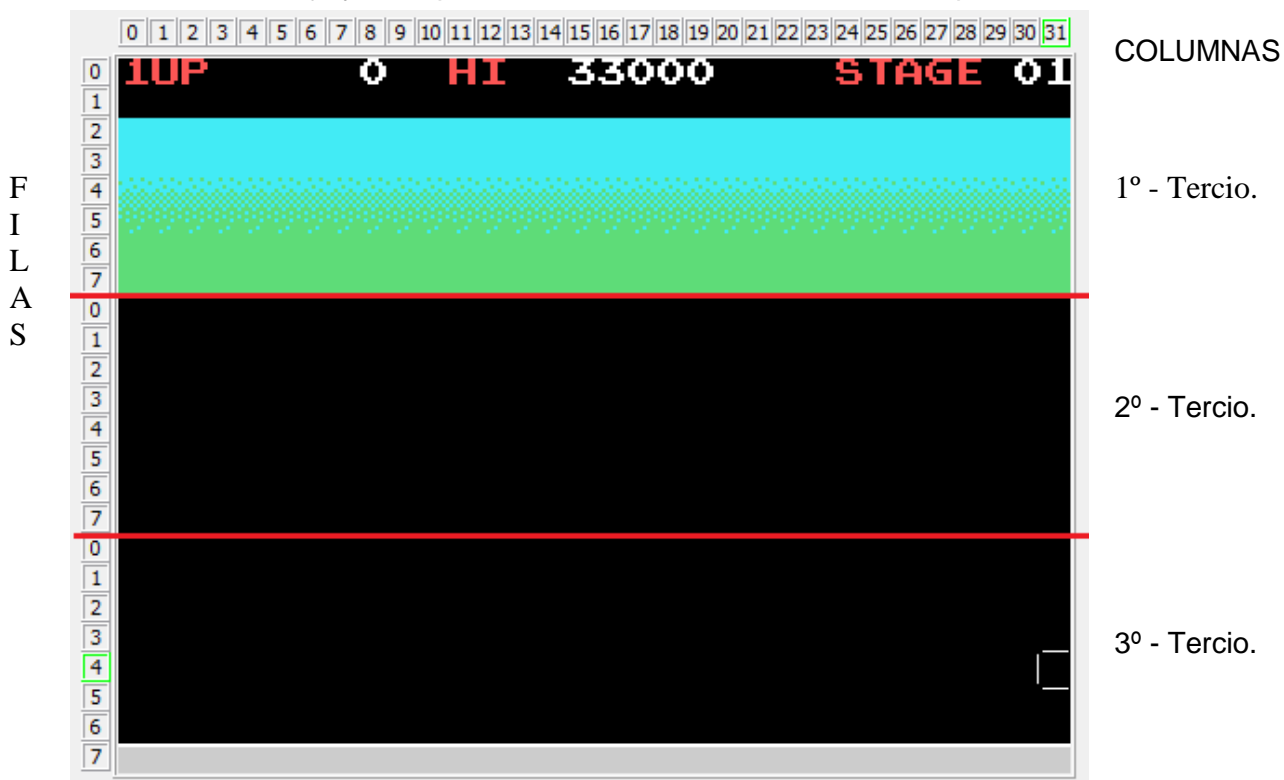


IMAGEN:

Banco de
Caracteres.

Banco 0

Este es el primer banco de caracteres que hemos colocado en la **Character Pattern Table** he puesto números dentro de cada CHR para que veas como se enumeran los CHRs empieza por el 0 y termina en el 255 de arriba abajo y de izquierda a derecha de la misma manera que nosotros leemos los textos.



Esta es la **Name Table** donde creamos las pantallas con los CHRs creando nuestro mapeado, he puesto unas líneas rojas para separar los 3 tercios, como solo he colocado un banco de caracteres solo podemos construir la pantalla en el primer tercio, para poder utilizar los otros 2 tercios tendríamos que colocar CHRs en los otros 2 bancos restantes y a partir de ahí podemos construir la pantalla completa.

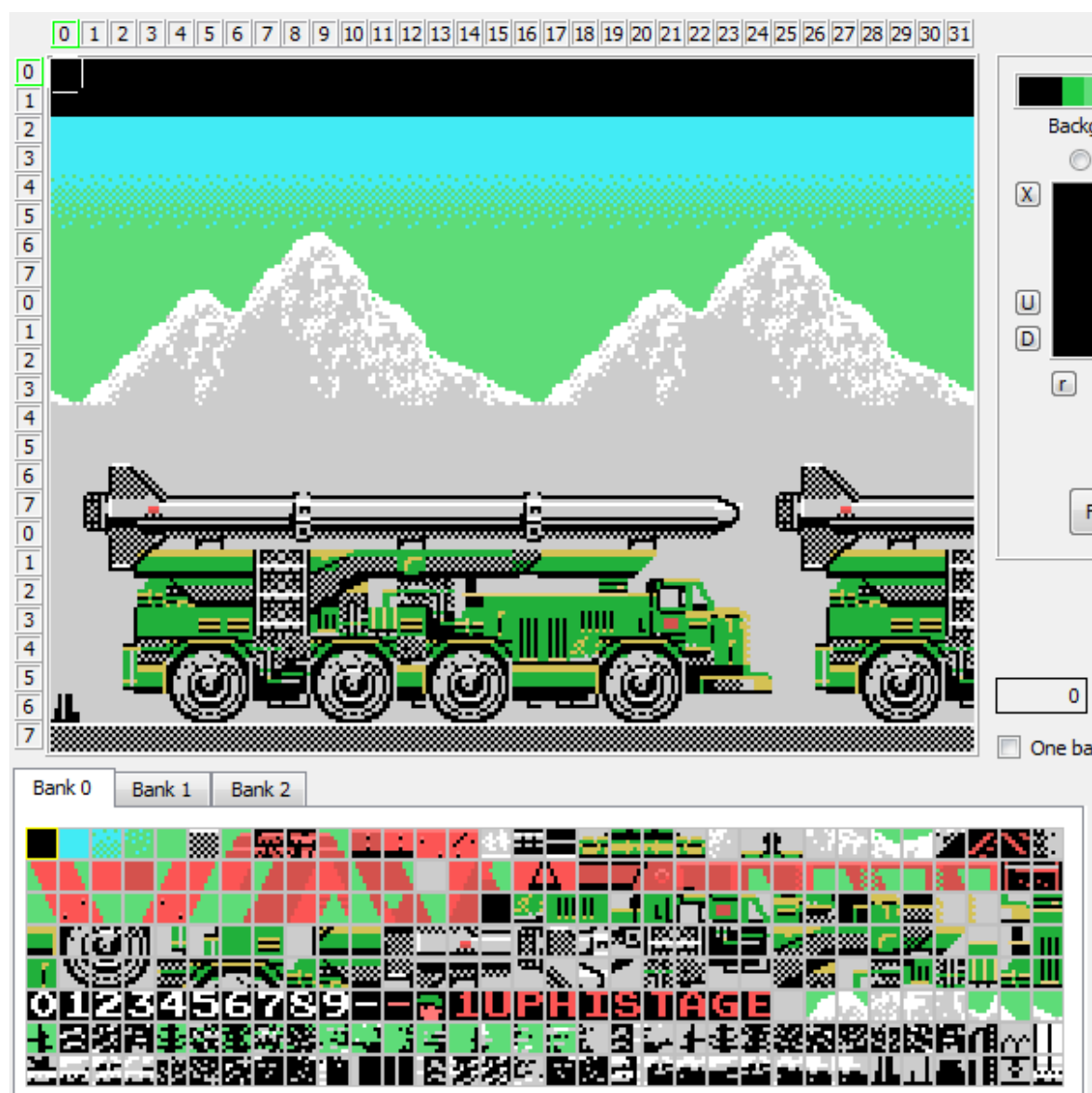
Ahora te enseño como se construye esta pantalla: Fíjate en la **columna 0-fila 0** hay un numero 1 Fíjate en la imagen de arriba donde tenemos el banco de CHRs en qué posición tenemos ese número, es el **CHR nº 6** pues en la **Name Table** en la **Columna 0 - fila 0** escribiríamos un **6**



La primera pantalla se construiría de la siguiente manera en la **columna 0 – fila 0** colocamos el **6** comprueba tu mismo viendo esta tabla y las 2 imágenes de arriba para que lo comprendas mejor.

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00	06	07	08	00	00	00	00	00	09	00	00	10	11	00	00	12	12	09	09	09	00	00	00	00	14	15	16	17	00	00	09	12
01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
02	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
03	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
04	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02
05	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03
06	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04
07	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04

Espero que todo lo hayas entendido a la perfección sino es así repásalo todas las veces que lo necesites ya que es muy importante que comprendas esta parte. Como puedes ver en las pantallas del mapeado los datos se repiten mucho y son fácilmente comprimibles ocupando muy poco espacio.



Aquí puedes ver una pantalla completa con los 3 bancos de CHRs en uso y como con pequeños CHRs se pueden construir unas pantallas con gráficos grandes. Imagen del nMSXtiles.

RESUMEN

Character Pattern Table: Ocupa 6Kb en la VRAM desde **0000h** a **17FFh** Abreviado **CHRTBL**
Aquí es donde situamos los CHRs que después usaremos para crear las pantallas.

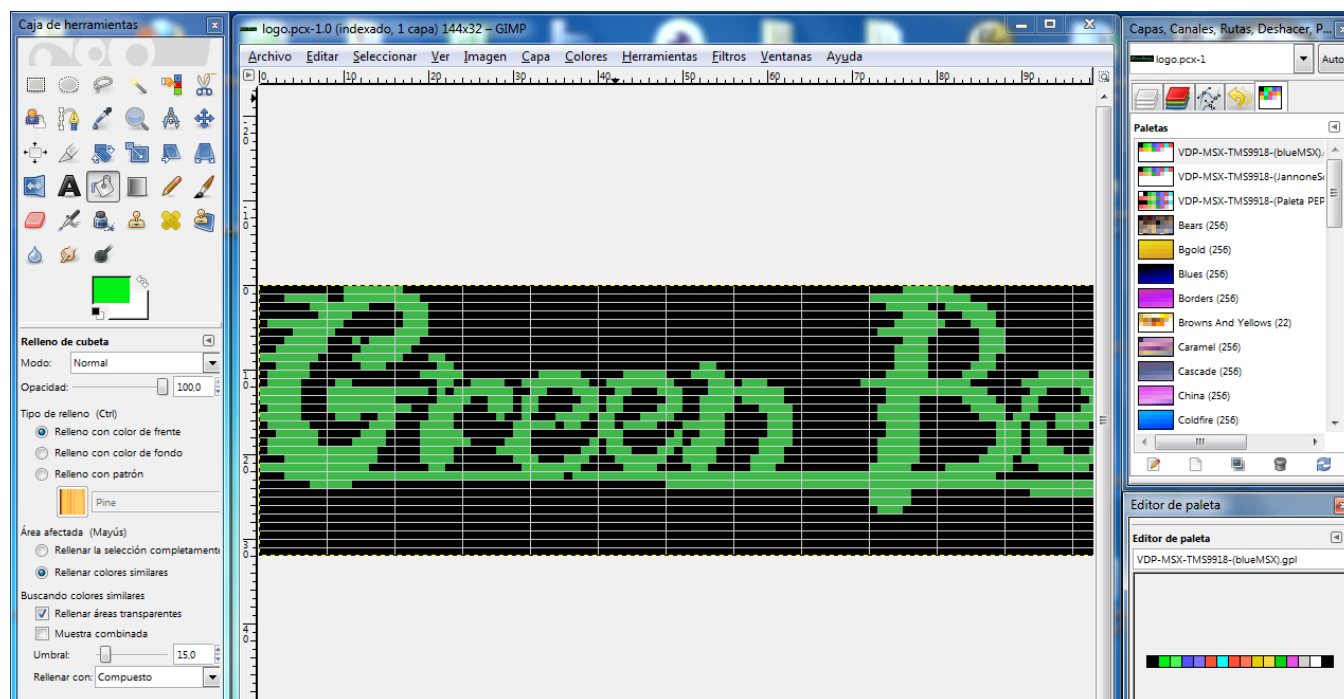
Colour Table: Ocupa 6Kb en la VRAM desde **2000h** a **37FFh** Abreviado **CLRTBL**
Aquí es donde colocamos los colores que tendrán los CHRs que tenemos en la CHRTBL

Name Table: Ocupa 768 Bytes en la VRAM desde **1800h** a **1CFFh** Abreviado **NAMTBL**
Aquí es donde situamos el mapeado con los números de CHRs que generan las pantallas.

Todos estos datos los usaremos después en el Código [Ensamblador](#) del [Hola Mundo Grafico](#).

Queda la **Sprite Pattern Table - SPRTBL** y la **Sprite Attribute Table - SPRATR** que veremos más adelante en otra entrega del tutorial dedicada al mundo de los Sprites.

Lo primero que necesitamos para empezar es un programa de dibujo que nos permita realizar los gráficos que después incorporaremos a nuestro código. Hay muchas opciones donde escoger yo personalmente me quede con el GIMP ya que es un programa gratuito tanto para Windows como Linux, muy completo. Página oficial para descargar el programa. <http://www.gimp.org> Bájatelo e instálalo.



Para configurar el GIMP de manera correcta para crear gráficos para el MSX, tenemos que configurar la resolución, la paleta de colores, y un Grid apropiado y grabar el fichero de imagen en formato PCX para después utilizarlo con otras herramientas. Fantástico manual de aOrante que me permito reproducir.

2ª Parte:

aOrante-----

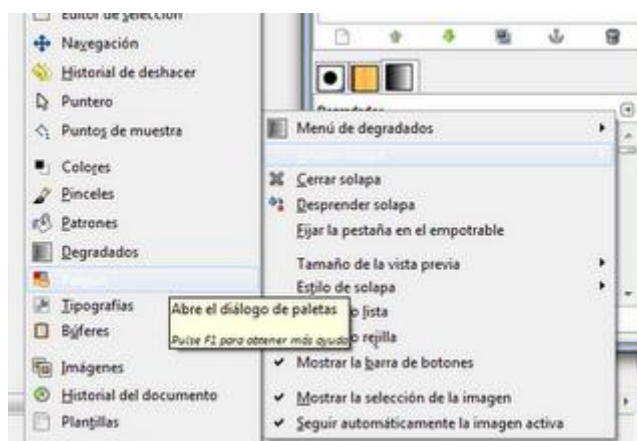
Una vez que tengamos abierto el programa, lo primero que haremos es crear una nueva imagen y para ello nos iremos al menú "Archivo" de la ventana principal y accionaremos la opción "Nuevo". Podemos acceder más rápidamente pulsando [CTRL]+[N]. Se nos mostrará una ventana donde nos pedirá el tamaño de la imagen y que le daremos 256 de anchura y 192 de altura.



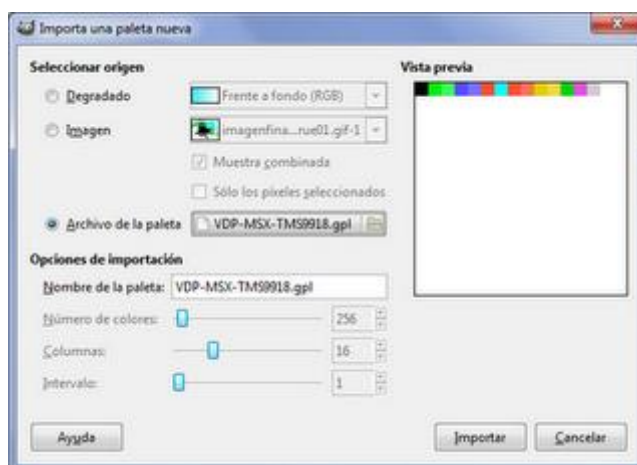
NOTA: Si vamos a utilizar a menudo Gimp para dibujar en este formato, podemos crear una plantilla desde "Archivo>Guardar como plantilla". Cuando empecemos una imagen, solo tendremos que indicar nuestra plantilla, desde el selector que se encuentra en la parte superior de la ventana para crear una nueva imagen.

El siguiente paso, será indicarle la cantidad de colores y la paleta que usaremos. Para nuestro caso necesitaremos que sea de tipo indexado de 16 colores con la paleta del VDP de los MSX. Para ello, necesitaremos disponer de la paleta. Podemos crearla nosotros desde "Ventanas>Diálogos empotrables>Paletas", pero para simplificaros la tarea he creado dos que podéis descargar desde este artículo. Una es la paleta que utilizan los emuladores blueMSX/OpenMSX y la otra es la del MSX Screen Conversor de Jannone, útil para cuando utilicemos esta aplicación. (Las paletas están al final para bajar)

Para cargarla, antes recomiendo los siguientes pasos para cambiar la configuración de la ventana de diálogos empotrables (la que por defecto se muestra a la derecha). Primero añadiremos una nueva solapa con el gestor de paletas. Hay dos formas, abrirla desde el menú "*Ventanas>Diálogos empotrables>Paletas*" y arrastrándola directamente a la zona inferior de la ventana. La segunda opción será pulsando ese pequeño botón con una flecha apuntando a la izquierda, situado en la parte superior derecha donde se encuentran las solapas. Se nos mostrará un menú y seleccionaremos la opción añadir solapa, donde encontraremos una llamada "*Paleta*".

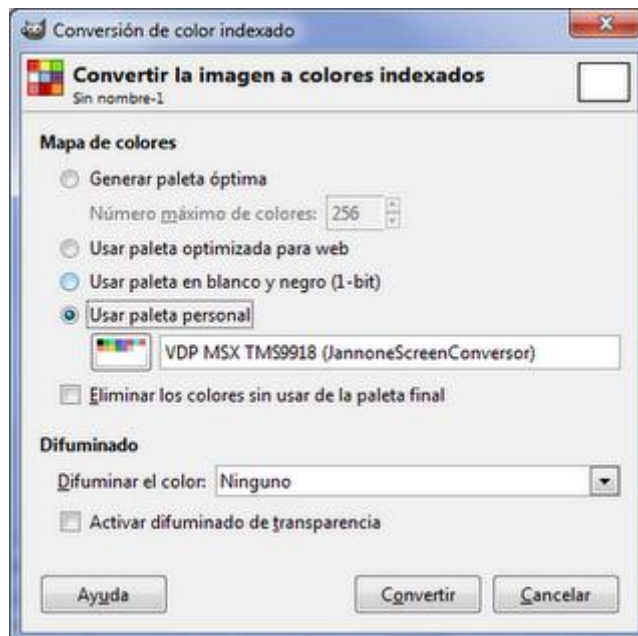


Una vez la tengamos insertada, pulsaremos de nuevo el pequeño botón, para acceder a la primera opción "*Menú de paletas*" y desde este seleccionaremos la opción "*Importar paleta...*". Luego utilizaremos la opción "*Archivo de la paleta*", para cargar la paleta.

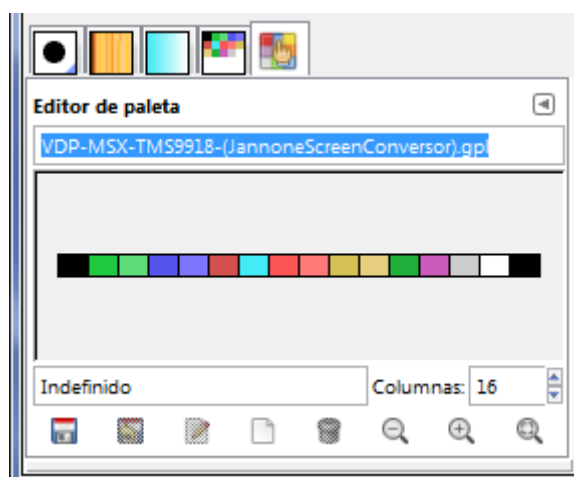


Ahora que ya tenemos la paleta, ya podemos indicar el tipo de imagen. Seleccionaremos "*Imagen>Modo>Indexado*". Se nos mostrará una ventana, Donde activaremos la opción "Usar paleta personal" y pulsaremos en el botón que hay en la línea inmediata, para seleccionar la paleta.

Importante: Inhabilitaremos el control que indica "*Eliminar los colores sin usar de la paleta final*" ya que de lo contrario, nos será imposible pintar en la pantalla.



El siguiente paso será poder acceder a nuestra paleta de colores y para ello, tendremos que añadir una solapa con el editor de paleta. Esto se consigue situándonos con el cursor encima de nuestra paleta y con el botón derecho abriremos el menú contextual donde pulsaremos sobre la opción de *"Editar paleta"*. Se nos abrirá una ventana, con nuestros colores, que aconsejo incrustar junto a la solapa del selector de paletas.



NOTA: Las modificaciones de la interfaz de usuario de Gimp, se mantienen al iniciar una nueva sesión, por lo que no deberemos repetir estos pasos de configuración, a excepción de la paleta, que se cambia por la "default".

Para ayudarnos a respetar las características del modo de pantalla del Screen2 (2 colores cada 8 pixeles), nos ayudaremos utilizando la rejilla. Antes de activarla, modificaremos su configuración mediante la opción *"Imagen>Configurar la rejilla"*. Se nos mostrará una ventana donde cambiaremos los valores de *"Espaciado"*, primeramente pulsando el icono de cadena, para inhabilitar la proporción, para luego indicar en la *"Anchura"* 8 y en *"Altura"* 1. Ahora toca activarla. Nos iremos al menú *"Ver"* y activaremos la opción *"Mostrar Rejilla"*.



Para poder ver correctamente la rejilla, tendremos que trabajar visualizando la imagen con un mínimo de un 400%. Esto se ajusta desde "*Ver>Ampliación*", o en el marco inferior de la ventana principal. Otra forma de trabajar será utilizando una rejilla de 8x8.



Ahora ya podemos dibujar. Una vez terminado y guardado el fichero, lo siguiente es tratar esos datos y eso lo explicaremos más adelante.

Paleta para Gimp (basada en la del BlueMSX modificada por PEPE)

Esta paleta la tenéis en el fichero grafico.rar de este tutorial.

Fichero [VDP-MSX-TMS9918-\(Paleta PEPE\).gpl](#)

aOrante-----

En el manual de aOrante habla de crear una pantalla de presentación de 256x192 pero podemos crear el tamaño que queramos dependiendo del grafico o gráficos que vallamos a crear, como puedes ver en mi primera captura para crear el logo del Green Beret especifico una resolución de 144 x 32 ya que solo quiero crear ese grafico para incorporarlo al código.

Vamos con la creación del set de caracteres que vamos a utilizar para el hola mundo grafico.

	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0000 0	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	` 96	p 112
0001 1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
0010 2	STX 2	DC2 18	" 34	2 50	B 66	R 82	b 98	r 114
0011 3	ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
0100 4	EOT 4	DC4 20	\$ 36	4 52	D 68	T 84	d 100	t 116
0101 5	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
0110 6	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
0111 7	BEL 7	ETB 23	' 39	7 55	G 71	W 87	g 103	w 119
1000 8	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1001 9	HT 9	EM 25) 41	9 57	I 73	Y 89	i 105	y 121
1010 A	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1011 B	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
1100 C	FF 12	FS 28	` 44	< 60	L 76	\ 92	l 108	 124
1101 D	CR 13	GS 29	- 45	= 61	M 77] 93	m 109	} 125
1110 E	SO 14	RS 30	. 46	> 62	N 78	^ 94	n 110	~ 126
1111 F	SI 15	US 31	/ 47	? 63	O 79	- 95	o 111	DEL 127

Tabla del código ASCII 7 bits

Según la tabla [ASCII](#) tienes que crear el set de caracteres o letras ciñéndote a su normativa.

Hay que empezar con el carácter 32 te lo remarco en color verde que es el espacio en blanco, y terminarlo en el carácter 127. Como veras en la imagen del set de caracteres que he dibujado a mano con el Gimp el orden de cada carácter esta creado según la normativa de la tabla ASCII.

32–Espacio, 33–Admiración, 34–Comillas, 35–Hash, 36–Dólar, 37–Porciento, Etc. Etc. hasta el 127.

Como puedes ver aquí cada letra se corresponde con un número de Carácter. De tal manera que cuando escribimos una letra "A" Mayúsculas remarcada en verde nuestro ordenador coloca un 65.

La explicación de crearlo de esta manera es porque después en nuestro código en ensamblador, pondremos los textos que vamos a usar en el tutorial en ASCII.

A la hora de situar los CHRs de nuestro set de letras en la VRAM empezaremos colocando el primer CHR en la posición 32 de la CHRTBL, para que coincida con la numeración que tienen en la tabla ASCII, de esta manera colocando en la NAMTBL un texto en ASCII coincidirá con el mismo número de carácter que tiene en la [Character Pattern Table - CHRTBL](#).

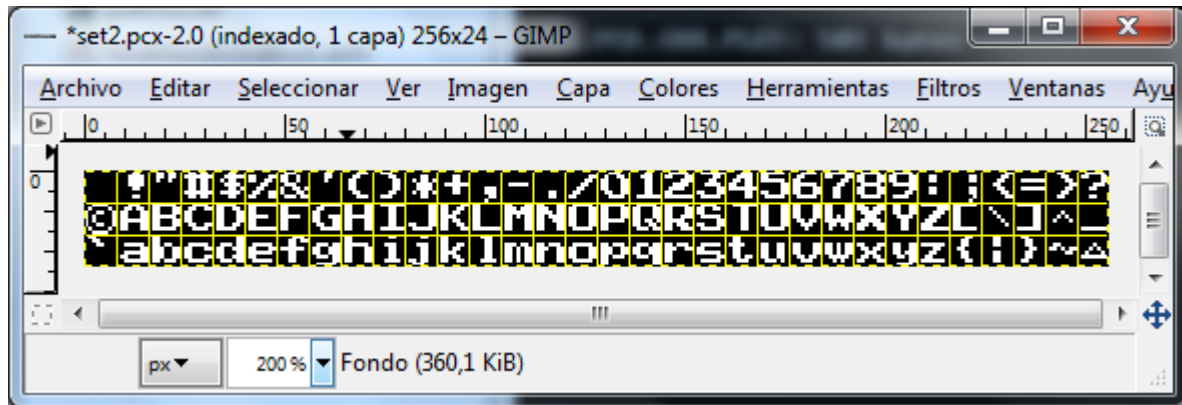
Como nuestro primer grafico es el eSPacio en el set de caracteres que hemos creado y su ASCII es el 32 es en este número donde tenemos que empezar a colocar los CHRs en la CHRTBL en la VRAM.


```
!"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{~
```

Estos son los 95 caracteres imprimibles ASCII en su orden correcto empieza con el carácter 20h o 32 que es el espacio (SP) seguido de símbolos con los números el alfabeto en Mayúsculas y el alfabeto en Minúsculas con algún que otro símbolo entremezclado.

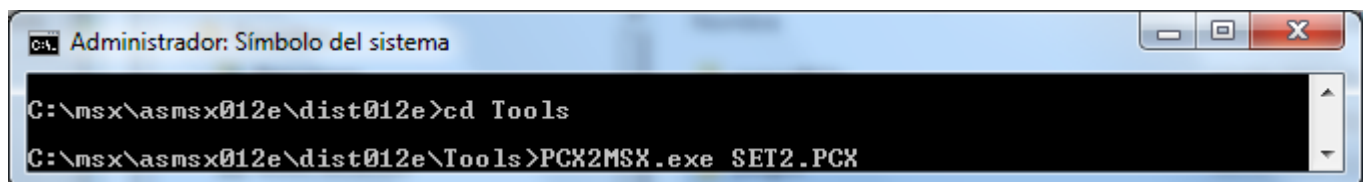
Aquí tienes el fichero [graficos.rar](#) con los ejemplos para este tutorial. Incluidos los gráficos código fuente y demás programas.

<http://www.megaupload.com/?d=7DWCTVD2>



Aquí puedes ver como he dibujado el set de caracteres con el GIMP, puedes crearte el tuyo propio o abrir el fichero [SET2.PCX](#) que yo he realizado para el ejemplo. (Aunque es mejor que vayas haciendo algo con el Gimp para que te vayas familiarizando.) He creado una nueva imagen con el tamaño de 256 que es el ancho total de la pantalla en screen2 por 24 de alto que son las 3 líneas de caracteres que necesito, y el Grid de 8x8 pixeles. (El resultado el que ves en la foto. Nota: no empleo color. Ya que este se puede poner por código mas tarde.) Fíjate como he creado el set de caracteres por si te animas a crear el tuyo propio, no has de usar los 8x8 pixeles en total deja una línea libre debajo y otra a la izquierda, para que cuando pintes letras seguidas estas no se toquen unas con otras y se mezclen.

Vamos con otra parte muy importante. Ya tenemos los gráficos en un fichero de imagen, como los incorporo a nuestro código para después compilar con el ensamblador [asMSX](#). Todas estas funciones las vamos a realizar desde el [MS-DOS](#) o [Símbolo del sistema](#). Copia el fichero SET2.PCX a la ruta donde vayas a crear el código en mi caso en [C:\msx\asmsx012e\dist012e\Tools\](#) En el [pack-MSX.rar](#) de la primera parte del tutorial descomprimos el [asMSX 0.12e](#) en [C:\MSX](#) abre desde Windows [botón Inicio – Todos los programas – Accesorios – Símbolo del sistema](#) Ahora teclea, [cd msx](#) intro, [cd asmsx012e](#) intro, [cd dist012e](#) intro, [cd Tools](#) intro. Fíjate en la imagen que te pongo debajo de este texto.



Ahora vamos a convertir la imagen [PCX](#) a fichero [BINARIO](#), para posteriormente comprimir los datos con el [compresor](#) y con el programa [binDB.exe](#) convertir el fichero [BINARIO](#) a [TEXTO](#) en formato [DB's](#) para nuestro ensamblador. (La necesidad de comprimir los datos, el espacio en ROM puede parecer muy grande pero los gráficos se llevan el 60% del tamaño de nuestra [ROM](#)) Estas 3 herramientas nos las proporciona nuestro querido maestro [E. Robsy](#), pero en mi caso comprimo los datos con [PLETTER](#) del grupo [XL2S Entertainment](#) ya que la compresión que realiza es muchísimo más intensa que la que realiza el [RLE](#). Teclea [PCX2MSX.exe SET2.PCX](#). Pulsamos intro.

(Tú puedes utilizar el método de compresión que prefieras, como [BITBUSTER](#), [MSX-o.-Mizer](#) etc. Lo que realmente hay que valorar es la velocidad, que la compresión sea buena y rutina con pocos bytes)

```









C:\msx\asmsx012e\dist012e>cd Tools
C:\msx\asmsx012e\dist012e\Tools>PCX2MSX.exe SET2.PCX

PCX2MSX v.0.10. PCX files to TMS9918 format. Edward A. Robsy Petrus [25/12/2004]
e-mail: edward@robsy.net - Web: http://www.robsy.net - (c) Karoshi Corp., 2004

-----
Original size: 256x24 pixels
TMS9918 size: 256x24 pixels
32x3=96 converted blocks
C:\msx\asmsx012e\dist012e\Tools>

```

Aquí tienes el resultado de que la operación se ha realizado con éxito, cuando trabajemos con gráficos o CHRs que tengan color presta atención al siguiente **ERROR Color collision at line (X,Y)** Esto quiere decir que hemos usado más de 2 colores por byte, Indicándote en que Línea y Columna esta el error.

	binDB.exe	29/12/2004 23:06	Aplicación	Aquí puedes ver cómo ha generado los dos ficheros binarios. SET2.PCX.chr que contiene los caracteres y SET2.PCX.clr que contiene los colores de los caracteres que no vamos a utilizar.
	MSXwav.exe	24/08/2004 20:51	Aplicación	
	PCX2MSX.exe	29/12/2004 23:31	Aplicación	
	PCX2MSXi.exe	29/12/2004 23:36	Aplicación	
	RLEpack.exe	29/11/2004 19:26	Aplicación	
	set2.pcx	17/09/2011 13:55	PhotoshopElemen...	
	SET2.PCX.chr	17/09/2011 15:55	Archivo CHR	
	SET2.PCX.clr	17/09/2011 15:55	Archivo CLR	

Ahora vamos a comprimir el fichero **BINARIO SET2.PCX.CHR** que son las letras de nuestro set.

```

C:\msx\asmsx012e\dist012e\Tools>RLEpack.exe SET2.PCX.CHR

RLEpack v.0.10. Run-length encode packer. Edward A. Robsy Petrus [29/11/2004]
e-mail: edward@robsy.net - Web: http://www.robsy.net - (c) Karoshi Corp., 2004

-----
SET2.PCX.CHR: 768 bytes
SET2.PCX.CHR.rle: 707 bytes [92%]
C:\msx\asmsx012e\dist012e\Tools>

```

Aquí puedes observar que solo ha reducido su tamaño un **8%** ya que el algoritmo de compresión del RLE es muy pobre y comprime muy poco los datos. Esto solo te lo muestro para que lo veas.

Ahora para que veas la diferencia voy a comprimir con PLETTER Ver. 5c1- usando **pletter.exe** de XL2S En los ejemplos de este tutorial en el **graficos.rar** dentro esta el fichero **pletter5c1.rar** descomprímelo y copia el fichero **pletter.exe** en el directorio de trabajo en **C:\msx\asmsx012e\dist012e\Tools**

```

C:\Windows\system32\cmd.exe

C:\msx\asmsx012e\dist012e\Tools>pletter.exe SET2.PCX.CHR SET2.PCX.CHR.PLET
Pletter v0.5c1 - www.xl2s.tk
..... SET2.PCX.CHR.PLET: 768 -> 503

C:\msx\asmsx012e\dist012e\Tools>_

```

Teclea **pletter.exe SET2.PCX.CHR SET2.PCX.CHR.PLET**. Aquí puedes ver la diferencia en **RLE** los **768 bytes** se quedan en **707 bytes** mientras que con **PLETTER** los **768 bytes** se quedan en **503 Bytes**.

binDB.exe	29/12/2004 23:06	Aplicación
BITpack.exe	24/11/2003 13:12	Aplicación
MSX-O-Mizer.exe	12/05/2008 20:59	Aplicación
MSXwav.exe	24/08/2004 20:51	Aplicación
PCX2MSX.exe	29/12/2004 23:31	Aplicación
PCX2MSXi.exe	29/12/2004 23:36	Aplicación
pletter.exe	28/02/2008 10:47	Aplicación
RLEpack.exe	29/11/2004 19:26	Aplicación
set2.pcx	01/10/2011 23:51	PhotoshopElemen...
set2.pcx.chr	05/10/2011 22:26	Archivo CHR
SET2.PCX.CHR.PLET	05/10/2011 22:58	Archivo PLET
set2.pcx.chr.rle	05/10/2011 22:31	RLE File
set2.pcx.clr	05/10/2011 22:26	Archivo CLR

Hemos Ganado 265 Bytes en la compresión. Solo en esto.

Al ejecutar el [RLEpack.exe](#) Nos genera un nuevo fichero acabado en [.rle](#)

Mientras que en [pletter](#) le decimos el nombre del fichero a comprimir **SET2.PCX.CHR** y el nombre que queremos darle al fichero comprimido yo le doy el mismo nombre pero le añado la extensión **.PLET** generándonos un nuevo fichero **SET2.PCX.CHR.PLET**

Vamos con el **.PLET** que es el de nuestro tutorial.

Ahora vamos a convertir el fichero **SET2.PCX.CHR.PLET** que ya es un binario comprimido a texto en formato **DB's** para incorporarlo a nuestro nuevo proyecto. Teclea **binDB.exe SET2.PCX.CHR.PLET**

```

C:\Windows\system32\cmd.exe

C:\msx\asmsx012e\dist012e\Tools>binDB.exe SET2.PCX.CHR.PLET

binDB v.0.10. binary to assembler DBs. Edward A. Robsy Petrus [29/11/2004]
e-mail: edward@robsy.net - Web: http://www.robsy.net - (c) Karoshi Corp., 2004

SET2.PCX.CHR.PLET: 503 bytes

C:\msx\asmsx012e\dist012e\Tools>

```

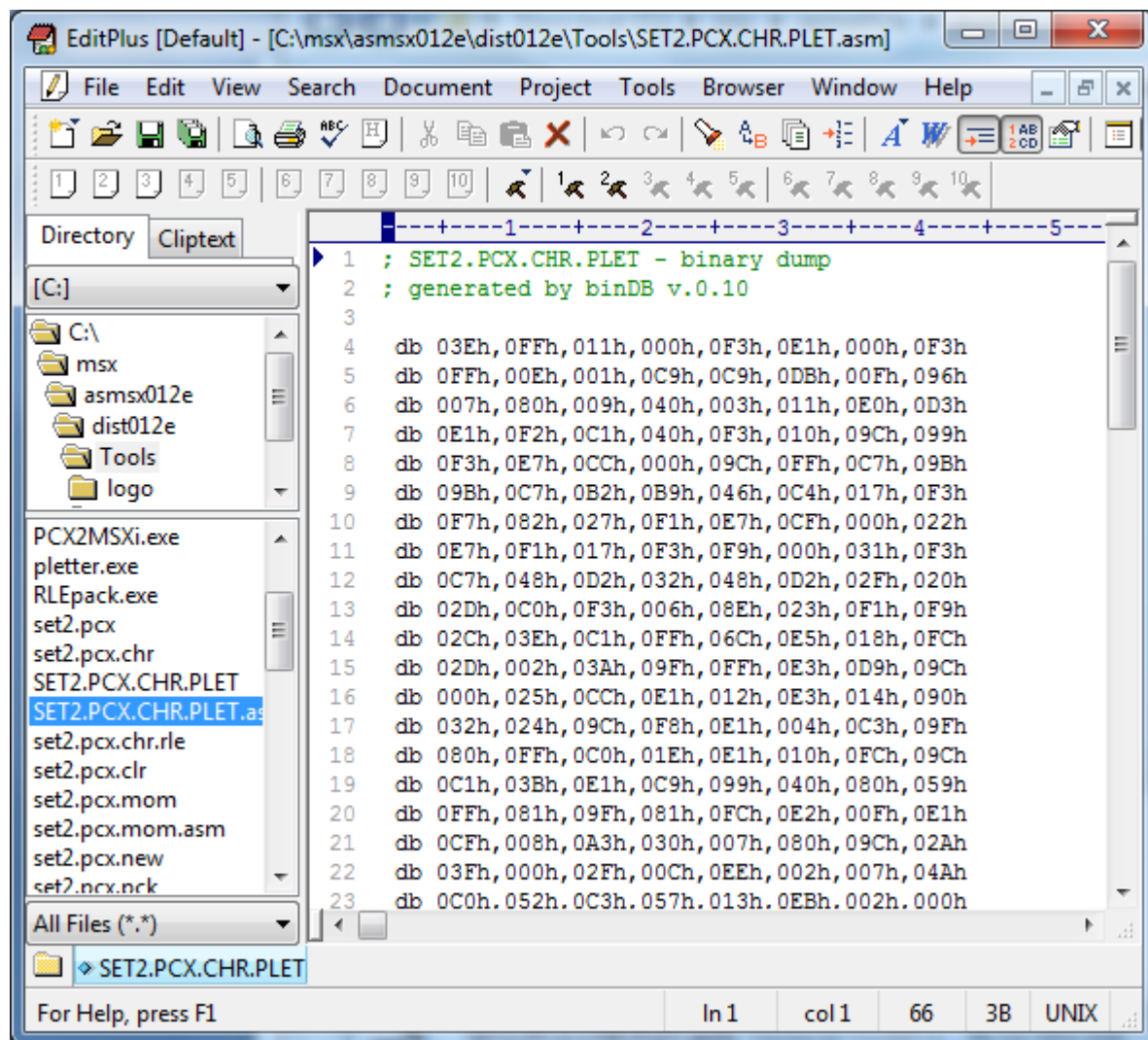
Este es el resultado final ahora tenemos un nuevo fichero llamado **SET2.PCX.CHR.PLET.asm**

binDB.exe	29/12/2004 23:06	Aplicación
BITpack.exe	24/11/2003 13:12	Aplicación
MSX-O-Mizer.exe	12/05/2008 20:59	Aplicación
MSXwav.exe	24/08/2004 20:51	Aplicación
PCX2MSX.exe	29/12/2004 23:31	Aplicación
PCX2MSXi.exe	29/12/2004 23:36	Aplicación
pletter.exe	28/02/2008 10:47	Aplicación
RLEpack.exe	29/11/2004 19:26	Aplicación
set2.pcx	01/10/2011 23:51	PhotoshopElemen...
set2.pcx.chr	05/10/2011 22:26	Archivo CHR
SET2.PCX.CHR.PLET	05/10/2011 22:58	Archivo PLET
SET2.PCX.CHR.PLET.asm	06/10/2011 0:41	EditPlus asm Z80 (...)
set2.pcx.chr.rle	05/10/2011 22:31	RLE File
set2.pcx.clr	05/10/2011 22:26	Archivo CLR

Aquí puedes ver el fichero acaba en **.asm** y está asociado para que se abra con el **EditPlus**.

Pulsa doble clic sobre el fichero y veras que se abre en nuestro editor listo para copiarlo y pegarlo a nuestro nuevo proyecto.

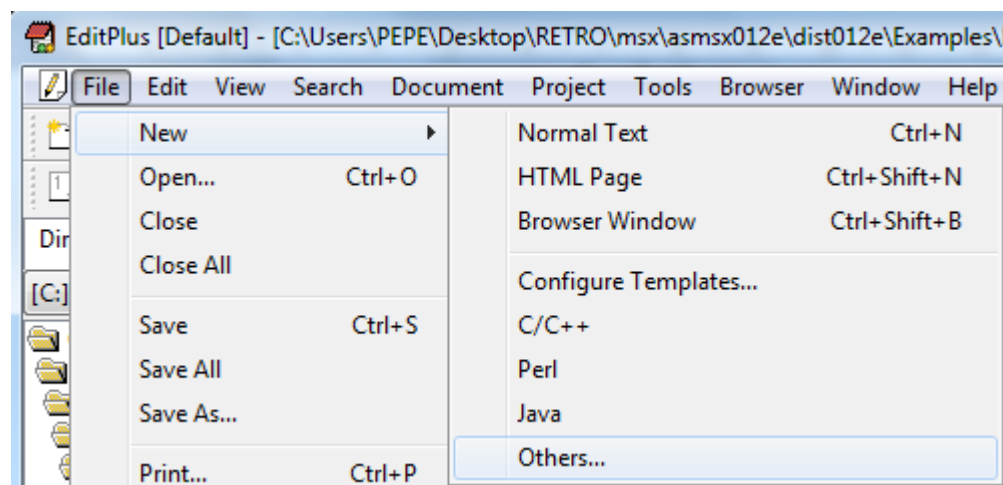
El tema de tener tantas extensiones es para saber de qué fichero de imagen parte todo el proceso. SET2.PCX es la IMAGEN .CHR que son los caracteres, .PLET es el fichero ya comprimido con pletter y .asm es el fichero en formato DB's listo para nuestro ensamblador.



Aquí tienes el set de caracteres en formato DB's para el ensamblador, esto son los bytes de los CHR's comprimidos que lo forman, y que veremos más adelante en este tutorial como tratarlos.

Tercera parte:

El código del Hola Mundo Grafico. Te atreves a empezar tu mismo la primera parte del código en ensamblador? Ya sabes una ROM, etc etc... como vimos en el primer y segundo manual.



Pulsa en el menú
[File – New – Others...](#) -

[asm Z80](#) y botón OK

Después en [File – Save As...](#) y lo guardas en el directorio que quieras y con el nombre que quieras. Yo le pongo [HolaMundoGrafico1.asm](#)

Seguimos teniendo el grafico en otra pestaña.

Vamos con el código del Hola Mundo Grafico

Fichero. "HolaMundoGrafico1.asm" está dentro del fichero **graficos.rar**

```
;-----
; Nombre de nuestro programa
; Hola Mundo Grafico - 24/09/2011
; Versión 1
;-----

;-----
; CONTANTES
;-----
; No definimos ninguna constante

;-----
; VARIABLES DEL SISTEMA
;-----
; Aquí definimos con nombres las direcciones en VRAM como te he explicado en la teoría de la primera parte del tutorial.
; Direcciones de la VRAM
        CHRTBL      equ    0000h    ; Tabla de caracteres
        NAMTBL      equ    1800h    ; Tabla de Nombres
        CLRTBL      equ    2000h    ; Tabla del color de los caracteres
        SPRATR      equ    1B00h    ; Tabla de los atributos de los sprites
        SPRTBL      equ    3800h    ; Tabla de Sprites
; Variables del Sistema MSX
        CLIKSW      equ    F3DBh    ; Keyboard click sound
        FORCLR      equ    F3E9h    ; Foreground colour

;-----
; DIRECTIVAS PARA EL ENSAMBLADOR ( asMSX )
;-----
        .bios        ; Definir Nombres de las llamadas a la BIOS
        .page 2      ; Definir la dirección del código irá en 8000h
        .rom         ; esto es para indicar que crearemos una ROM
        .start INICIO ; Inicio del Código de nuestro Programa
; Seguir la norma del Standard MSX para una ROM
        dw 0,0,0,0,0 ; 12 Ceros

;-----
; INICIO DEL PROGRAMA
;-----
INICIO:
        call    INIT_MODE_SCx ; iniciar el modo de pantalla x
        call    INIT_GRAFICOS ; colocar los gráficos en VRAM

FIN:
        jp      FIN           ; esto es como 100 goto 100 para que se quede en un bucle sin fin.

;-----

; Todo esto lo hemos visto en el tutorial anterior así que no necesita muchas mas explicaciones
; La única diferencia es que aquí usamos el SCREEN 2, por eso activo la opción - call INIGRP
; Esta rutina os sirve para activar cualquiera de estos modos descritos en los textos a modo de comentario.

;-----
; INICIALIZA EL MODO DE PANTALLA Y COLORES
;-----
; BASIC: COLOR 15,1,1
; Establecer los colores

INIT_MODE_SCx:
        ld      hl,FORCLR      ; Variable del Sistema
        ld      [hl],15        ; Color del primer plano 15=blanco
        inc     hl              ; FORCLR+1
        ld      [hl],1         ; Color de fondo 1=negro
        inc     hl              ; FORCLR+2
        ld      [hl],1         ; Color del borde 1=negro
; call INITXT      ; BIOS set SCREEN 0
; call INIT32      ; BIOS set SCREEN 1
        call    INIGRP         ; BIOS set SCREEN 2
; call INIMLT      ; BIOS set SCREEN 3
;
; SCREEN 0 : texto de 40 x 24 con 2 colores
; SCREEN 1 : texto de 32 x 24 con 16 colores
; SCREEN 2 : gráficos de 256 x 192 pixeles con 16 colores
; SCREEN 3 : gráficos de 64 x 48 pixeles con 16 colores

        ret
;-----
```

```

;-----
; COLOCA LOS GRAFICOS EN LA VRAM
;-----
INIT_GRAFICOS:
; Esto lo realizamos para que no se vea nada en la pantalla
; Mientras colocamos los CHR's y los Colores en la VRAM
    call    DISSCR        ; BIOS deshabilitar la pantalla

; Esto es para quitar el sonido que emite el msx cuando se pulsa una tecla
    xor     a             ; ld a,0
    ld      [CLIKSW],a    ; Variable BIOS desactivar sonido teclas

; Borrar los 768 Bytes de la Name Table - NAMTBL en VRAM
; Originalmente la BIOS rellena la NAMTBL con los valores del 0 al 255 en cada tercio por eso lo relleno todo con 0
    ld      hl,NAMTBL     ; Dirección origen en VRAM
    ld      bc,768        ; nº de CHR's a rellenar
    xor     a             ; a=0 - Valor a rellenar
    call    FILVRM        ; BIOS -Fill block of VRAM with data byte ; Te explico más abajo el funcionamiento de FILVRM

```

Aquí entra en acción algo que todavía no te he explicado en anteriores tutoriales, esta parte es la encargada de colocar las letras que hemos creado, y que tenemos comprimidas en formato DB's en la VRAM, te pongo un fragmento debajo de este texto para que lo entiendas.

```

;-----
; SET2.PCX.CHR.PLET - binary dump
; generated by binDB v.0.10
;-----
SET2_PLET:
    db  03Eh,000h,011h,000h,018h,03Ch,000h,018h
    db  000h,00Eh,001h,06Ch,06Ch,048h,00Fh,096h
    db  007h,0FEh,009h,000h,003h,000h,030h,07Ch
    db  0B0h,078h,034h,0F8h,040h,030h,010h,0C6h
    .....

```

Este es el fichero generado por el programa binDB.exe
obtenido de la imagen PCX y comprimido con PLETTER
debes copiar el texto y pegarlo en tu código al final del todo
Ahora tienes que colocar una etiqueta para que apunte al
principio de los datos de nuestro set de caracteres.
Yo escribo. **SET2_PLET:**
Justo debajo de los comentarios del binDB.

Si no quieres que tu código en ensamblador sea tan grande como cuando creas un videojuego, puedes omitir crear el fichero que ves encima de este texto con el bindb.exe y colocar el fichero binario que has comprimido con el compresor directamente, de la siguiente manera.

```

;-----
; Set de caracteres BINario comprimido con PLETTER
SET2_PLET:
    .INCBIN        "SET2.PCX.CHR.PLET"
;-----

```

Este es el código que irá colocando los bytes ya descomprimidos de nuestros set de caracteres en la VRAM en la **Character Pattern Table**.

```

; Colocar los gráficos de las letras en VRAM en la CHRTBL
    ld      hl,SET2_PLET    ; set de CHR's de las letras
    ld      de,CHRTBL+32*8  ; Empezar en el CHR 32
    call    DEPLET          ; Descomprimir en VRAM

```

La rutina **DEPLET** está situada en nuestro código y tiene unos parámetros de entrada, que son los que te describo a continuación:
El registro **HL** tiene que apuntar a la dirección de memoria donde empiezan los bytes de nuestros datos comprimidos.
En el ejemplo que nos ocupa apunta a la dirección **SET2_PLET** que es donde están los bytes comprimidos del Set de Caracteres.

```

    ld      hl,SET2_PLET    ; set de CHR's de las letras

```

El registro **DE** tiene que apuntar a la dirección en VRAM donde queremos descomprimir los bytes.
Quiero empezar a colocar bytes en el **CHR 32** de la **CHRTBL** en VRAM por eso le sumo a CHRTBL, 32 x 8 bytes que tiene cada carácter.
La explicación de colocar el set de caracteres a partir del CHR 32 la explico con más detalle en la página 9 de este tutorial.

```

    ld      de,CHRTBL+(32*8) ; Empezar en el CHR 32

```

Esta es la llamada a la rutina que toma los bytes comprimidos y los descomprime en VRAM según los valores pasados en **HL** y **DE**.
Empieza a tomar bytes desde **SET2_PLET** los descomprime y los coloca en la **Character Pattern Table - CHRTBL** a partir del CHR 32.

```

    call    DEPLET          ; Descomprimir en VRAM

```

```

; Colocar el color de las letras en VRAM en la CLRTBL
    ld      hl,CLRTBL+(32*8) ; Empezar en el CHR 32
    ld      bc,32*24         ; numero de CHR's
    ld      a,0Fh            ; Valor a rellenar
    call    FILVRM           ; BIOS -Fill block of VRAM with data byte

```

La rutina **FILVRM** está situada en la BIOS y tiene unos parámetros de entrada, que son los que te describo a continuación:
El registro **HL** tiene que apuntar a la dirección de memoria de la VRAM donde queremos empezar a colocar los bytes.
En el ejemplo que nos ocupa queremos empezar a colocar bytes en el CHR 32 de la **Colour Table - CLRTBL**.

```

    ld      hl,CLRTBL+(32*8) ; Empezar en el CHR 32

```

En el registro **BC** le decimos cuantos son los bytes que hay que rellenar.
El set de caracteres ocupa 32 CHR's de ancho por 3 CHR's de alto, pero recuerda que cada CHR tiene de alto 8 bytes, entonces hay que multiplicar 3 x 8=24 bytes de altura, por eso en el registro **BC** pongo 32 x 24

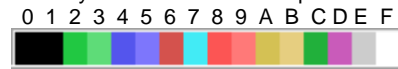
```

    ld      bc,32*24         ; numero de CHR's

```

En el registro **A** le decimos el byte que tiene que usar para rellenar toda la zona seleccionada.

En el ejemplo que nos ocupa el Color de la tinta es blanco F y el fondo negro 0, o lo que es lo mismo **0Fh** en hexa. 16 Colores de fondo y 16 colores de tinta, recuerda que los colores empiezan con el 0 y terminan en el 15 si miras la paleta del MSX veras que el blanco es el 15 y el negro el 0 - 15 en hexadecimal es F y el negro 0, resultado 0Fh si quieres cambiar colores ya sabes cómo tienes que hacerlo.



```
ld      a,0Fh          ; Valor a rellenar
```

Esta es la llamada a la rutina que empezara escribiendo el byte 0Fh en VRAM empezando en CLRTBL+256 y no parara hasta que se rellenen 32x24=768 Bytes que son las 3 filas de 32 CHR que ocupa el set de caracteres en la [Color Table - CLRTBL](#)

```
call    FILVRM          ; BIOS –Fill block of VRAM with data byte
```

; Colocar la cadena de texto en la pantalla
; LOCATE 2,6: PRINT "Hola Mundo Grafico"

```
ld      hl,TXT_HOLA      ; Dirección donde tenemos el texto  
ld      de,NAMTBTL+2*32+6 ; LOCATE 2,6 : Destino la NAMBTBL en VRAM  
ld      bc,18            ; Numero de CHR que tiene el texto  
call    LDIRVM           ; BIOS – Copy block to VRAM, from memory
```

La rutina **LDIRVM** está situada en la BIOS y tiene unos parámetros de entrada, que son los que te describo a continuación:

El registro **HL** tiene que apuntar a la dirección de memoria RAM-ROM donde tenemos los bytes que queremos llevar a la VRAM

En el ejemplo que nos ocupa donde tenemos la cadena de texto en ASCII que queremos llevar a la VRAM

```
ld      hl,TXT_HOLA      ; Dirección donde tenemos el texto
```

El registro **DE** tiene que apuntar a la dirección en VRAM donde queremos colocar los bytes que están en RAM-ROM

Quiero empezar a colocar bytes en la [Name Table – NAMTBTL](#) en la Fila 2 – Columna 6 de la pantalla

Por eso le sumo a la dirección. NAMTBTL la fila nº 2 x32 CHR que tiene cada fila y le añado los 6 de la Columna

```
ld      de,NAMTBTL+2*32+6 ; LOCATE 2,6
```

En el registro **BC** le decimos el número de bytes que transferimos de RAM-ROM a VRAM

En el ejemplo que nos ocupa 18 bytes que son los CHR que ocupa el texto "Hola Mundo Grafico"=18 letras.

```
ld      bc,18            ; Numero de CHR que tiene el texto
```

Esta es la llamada a la rutina que empezara a trasferir 18 bytes desde la RAM-ROM hacia la VRAM concretamente en la NAMTBTL+70

En el ejemplo que nos ocupa el valor de los códigos ASCII del texto, los situara en la [Name Table - NAMTBTL](#)

```
call    LDIRVM           ; BIOS – Copy block to VRAM, from memory
```

; Esto lo realizamos para que se muestre de nuevo la pantalla.

```
call    ENASCR           ; BIOS habilitar la pantalla
```

; Salir de la rutina INIT GRAFICOS

```
ret
```

Vamos con el texto para nosotros es más fácil poner **db "HOLA MUNDO Grafico"** que tener que ver la tabla ASCII de la pagina 9 y mirar los valores que corresponden a cada letra, y tener que poner **db 72,79,76,65,32,77,85,78,68,79,32,71,114,97,102,105,99,111** pero esta labor la hace por nosotros el asMSX a la hora de compilar, ahora eso si en la NAMTBTL los valores que se escribirán serán los valores ASCII, repasa la pagina 9 de este tutorial que explica este apartado con más detalle.

; Cadena de Texto a visualizar en la pantalla

TXT_HOLA:

```
db      "HOLA MUNDO Grafico"
```

Esta es la rutina encargada de la descompresión de bytes en VRAM en mi caso yo he escogido el sistema de compresión PLETTER, pero el mercado hay un gran elenco de compresores-descompresores como RLE, BITBUSTER,MSX-o-Mizer etc. Tu puedes seleccionar el sistema de compresión que más se adapte a tus necesidades cambiando esta rutina por el código del sistema de compresión que tu hayas escogido, y comprimir el fichero binario de tu imagen con el compresor que te suministren.

Aquí tienes la rutina del descompresor Pletter v0.5b de RAM-ROM a VRAM directamente. Si quieres puedes estudiar su código para saber cómo funciona o simplemente utilizarla, ya te he explicado anteriormente que parámetros de entrada debes de proporcionarle a la rutina.

; Pletter v0.5b VRAM Depacker v1.1 - 16 Kb version

; HL = RAM/ROM source

; DE = VRAM destination

DEPLET:

```
di
```

; VRAM address setup

```
ld      a,e  
out     (099h),a  
ld      a,d  
or      040h  
out     (099h),a
```

```

; Initialization
    ld    a,[hl]
    inc   hl
    exx
    ld    de,0
    add   a,a
    inc   a
    rl    e
    add   a,a
    rl    e
    add   a,a
    rl    e
    rl    e
    ld    hl,modes
    add   hl,de
    ld    e,[hl]
    ld    ixl,e
    inc   hl
    ld    e,[hl]
    ld    ixh,e
    ld    e,1
    exx
    ld    iy,loop

; Main depack loop
literal: ld    c,098h
        outi
        inc   de
loop:   add   a,a
        call  z,getbit
        jr    nc,literal

; Compressed data
        exx
        ld    h,d
        ld    l,e
getlen: add   a,a
        call  z,getbitexx
        jr    nc,lenok
lus:   add   a,a
        call  z,getbitexx
        adc   hl,hl
        ret   c
        add   a,a
        call  z,getbitexx
        jr    nc,lenok
        add   a,a
        call  z,getbitexx
        adc   hl,hl
        jp    c,Depack_out
        add   a,a
        call  z,getbitexx
        jp    c,lus
lenok: inc   hl
        exx
        ld    c,[hl]
        inc   hl
        ld    b,0
        bit   7,c
        jp    z,offsok
        jp    ix

mode7: add   a,a
        call  z,getbit
        rl    b
mode6: add   a,a
        call  z,getbit
        rl    b
mode5: add   a,a
        call  z,getbit
        rl    b
mode4: add   a,a
        call  z,getbit
        rl    b
mode3: add   a,a
        call  z,getbit
        rl    b
mode2: add   a,a

```

```

call    z,getbit
rl      b
add     a,a
call    z,getbit
jr      nc,offsok
or      a
inc     b
res     7,c
offsok: inc     bc
        push   hl
        exx
        push   hl
        exx
        ld     l,e
        ld     h,d
        sbc    hl,bc
        pop    bc
        push   af
@@loop: ld     a,l
        out    (099h),a
        ld     a,h                ; VDP timing
        nop
        out    (099h),a          ; VDP timing
        nop
        in     a,(098h)
        ex     af,af
        ld     a,e                ; VDP timing
        nop
        out    (099h),a
        ld     a,d
        or     040h
        out    (099h),a
        ex     af,af              ; VDP timing
        nop
        out    (098h),a
        inc    de
        cpi    pe,@@loop
        jp     af
        pop    hl
        pop    iy
getbit: ld     a,[hl]
        inc    hl
        rla
        ret

getbitexx:
        exx
        ld     a,[hl]
        inc    hl
        exx
        rla
        ret

; Depacker exit
Depack_out:
        ei
        ret

modes:
        dw     offsok
        dw     mode2
        dw     mode3
        dw     mode4
        dw     mode5
        dw     mode6
        dw     mode7

```

Aquí en esta parte del código es donde tienes que pegar el código en formato DB's del set de caracteres que previamente has comprimido y convertido a formato DB's con el binDB.exe y que te he explicado anteriormente.

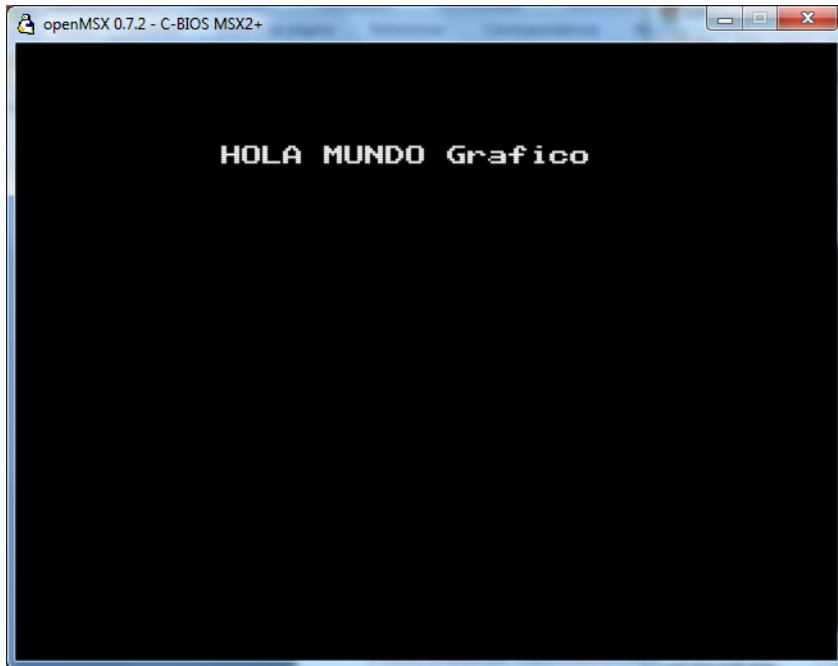
```
;-----  
; SET2.PCX.CHR.PLET - binary dump  
; generated by binDB v.0.10  
;-----
```

SET2_PLET:

```
db 03Eh,0FFh,011h,000h,0F3h,0E1h,000h,0F3h  
db 0FFh,00Eh,001h,0C9h,0C9h,0DBh,00Fh,096h  
db 007h,080h,009h,040h,003h,011h,0E0h,0D3h  
db 0E1h,0F2h,0C1h,040h,0F3h,010h,09Ch,099h  
db 0F3h,0E7h,0CCh,000h,09Ch,0FFh,0C7h,09Bh  
db 09Bh,0C7h,0B2h,0B9h,046h,0C4h,017h,0F3h  
db 0F7h,082h,027h,0F1h,0E7h,0CFh,000h,022h  
db 0E7h,0F1h,017h,0F3h,0F9h,000h,031h,0F3h  
db 0C7h,048h,0D2h,032h,048h,0D2h,02Fh,020h  
db 02Dh,0C0h,0F3h,006h,08Eh,023h,0F1h,0F9h  
db 02Ch,03Eh,0C1h,0FFh,06Ch,0E5h,018h,0FCh  
db 02Dh,002h,03Ah,09Fh,0FFh,0E3h,0D9h,09Ch  
db 000h,025h,0CCh,0E1h,012h,0E3h,014h,090h  
db 032h,024h,09Ch,0F8h,0E1h,004h,0C3h,09Fh  
db 080h,0FFh,0C0h,01Eh,0E1h,010h,0FCh,09Ch  
db 0C1h,03Bh,0E1h,0C9h,099h,040h,080h,059h  
db 0FFh,081h,09Fh,081h,0FCh,0E2h,00Fh,0E1h  
db 0CFh,008h,0A3h,030h,007h,080h,09Ch,02Ah  
db 03Fh,000h,02Fh,00Ch,0EEh,002h,007h,04Ah  
db 0C0h,052h,0C3h,057h,013h,0EBh,002h,000h  
db 023h,033h,002h,0FDh,0FBh,0A1h,066h,09Ah  
db 0FCh,095h,010h,05Bh,001h,075h,01Eh,00Ch  
db 012h,0C5h,037h,0FCh,0F1h,029h,000h,001h  
db 0C3h,0BDh,066h,05Eh,05Eh,066h,0BDh,015h  
db 0C3h,0E3h,0C9h,040h,053h,019h,0DFh,05Ch  
db 0D1h,002h,067h,0CCh,09Fh,060h,000h,097h  
db 083h,050h,099h,00Dh,0FAh,083h,0A4h,06Fh  
db 00Dh,083h,002h,080h,0F2h,007h,0B7h,010h  
db 0E0h,087h,098h,084h,0B7h,0E0h,096h,0E9h  
db 036h,046h,037h,0ECh,0C3h,0BFh,0FCh,0B0h  
db 000h,08Fh,082h,033h,093h,087h,093h,099h  
db 017h,075h,09Fh,000h,037h,00Eh,00Fh,088h  
db 080h,094h,027h,00Ah,09Ch,08Ch,084h,090h  
db 038h,007h,0B9h,07Fh,000h,0C1h,0D6h,06Fh  
db 06Dh,070h,04Fh,00Fh,092h,099h,0C4h,0E0h  
db 00Fh,098h,083h,091h,098h,081h,097h,099h  
db 09Fh,0C1h,0DBh,047h,02Fh,057h,0F3h,067h  
db 038h,02Fh,005h,0C9h,0E3h,072h,0F7h,007h  
db 094h,09Eh,098h,0DDh,057h,0C1h,012h,0E3h  
db 0C1h,088h,04Fh,0CCh,000h,070h,0E1h,027h  
db 080h,0F8h,0F1h,015h,0E3h,0C7h,08Fh,06Fh  
db 053h,0C8h,000h,081h,0F0h,002h,0DFh,0EFh  
db 0F7h,0FBh,0FDh,0FFh,00Fh,075h,0F9h,000h  
db 00Fh,017h,000h,0F7h,0EBh,035h,0D2h,000h  
db 027h,09Fh,0BCh,0BCh,00Bh,093h,06Eh,0C0h  
db 0C2h,081h,0AFh,093h,08Ch,038h,000h,091h  
db 00Fh,09Ch,09Fh,0C8h,06Fh,0CCh,021h,0C4h  
db 098h,091h,0C4h,0A5h,00Fh,053h,090h,08Fh  
db 0E4h,0E7h,081h,0E6h,0F7h,000h,0A3h,017h  
db 016h,0FCh,0C1h,038h,063h,092h,0CFh,0B9h  
db 0A7h,0C0h,0E1h,085h,007h,093h,0C7h,017h  
db 0A3h,080h,0A7h,00Dh,046h,017h,000h,089h  
db 094h,061h,000h,007h,091h,08Ch,0DDh,02Fh  
db 04Fh,0C7h,0CFh,057h,00Fh,06Eh,02Dh,08Eh  
db 04Fh,0FCh,00Fh,0E7h,0DFh,032h,01Fh,0F1h  
db 0ADh,01Fh,0DCh,087h,046h,002h,0F0h,031h  
db 00Fh,099h,000h,0C5h,099h,007h,036h,0A6h  
db 00Fh,01Dh,04Dh,0C1h,0EBh,00Fh,03Ah,00Dh  
db 0C9h,04Fh,080h,046h,0C8h,0E4h,0FCh,033h  
db 068h,081h,0D7h,08Ah,0E7h,004h,005h,0A3h  
db 002h,08Dh,08Eh,0ABh,003h,0ECh,00Ch,012h  
db 0A2h,027h,009h,094h,0F9h,005h,0D9h,087h  
db 05Fh,080h,005h,0FFh,0FFh,0FFh,0F8h
```

```
;-----  
O bien puedes quitar todos estos DB´s e incluir el fichero BINARIO directamente en el código de esta manera  
El fichero SET2.PCX.PLET Tiene que estar en el directorio donde tengas el código en ensamblador.
```

```
;-----  
; SET2.PCX.CHR.PLET  
; Set de caracteres en BINario comprimido con PLETTER  
;SET2_PET:  
; .INCBIN "SET2.PCX.CHR.PLET"  
;-----  
; FINAL DE NUESTRO CODIGO EN ENSAMBLADOR.  
;-----
```



Este es el resultado Final que veras si todo lo has realizado de manera correcta.



Como queda un poco soso, sin colorido y demás en la segunda parte del mundo de los gráficos vamos a ver todo este proceso.

La imagen de la izquierda veremos cómo hacerla en la 2ª entrega.

Os animáis a crearlo vosotros mismos, con todo lo que habéis aprendido en este tutorial, debería bastaros para que vosotros mismos podáis realizarlo.

Espero ver vuestros comentarios y vuestras capturas de pantallas, en los BLOG´s o FORO´s del tutorial.

Espero que haya sido de vuestro total agrado y nos vemos en el próximo tutorial. Donde explicaré como dar un colorido especial a nuestras letras desde código, como incorporar gráficos a nuestro Hola Mundo Grafico y todo lo relacionado con la creación de pantallas o lo que es lo mismo trabajar con la [NAMTBL](#) creando mapeados usando el [nMSXtiles](#).

José Vila Cuadrillero

"ES DETESTABLE ESA AVARICIA ESPIRITUAL QUE TIENEN, LOS QUE SABIENDO ALGO, NO PROCURAN LA TRANSMISION DE ESOS CONOCIMIENTOS."

Miguel de Unamuno

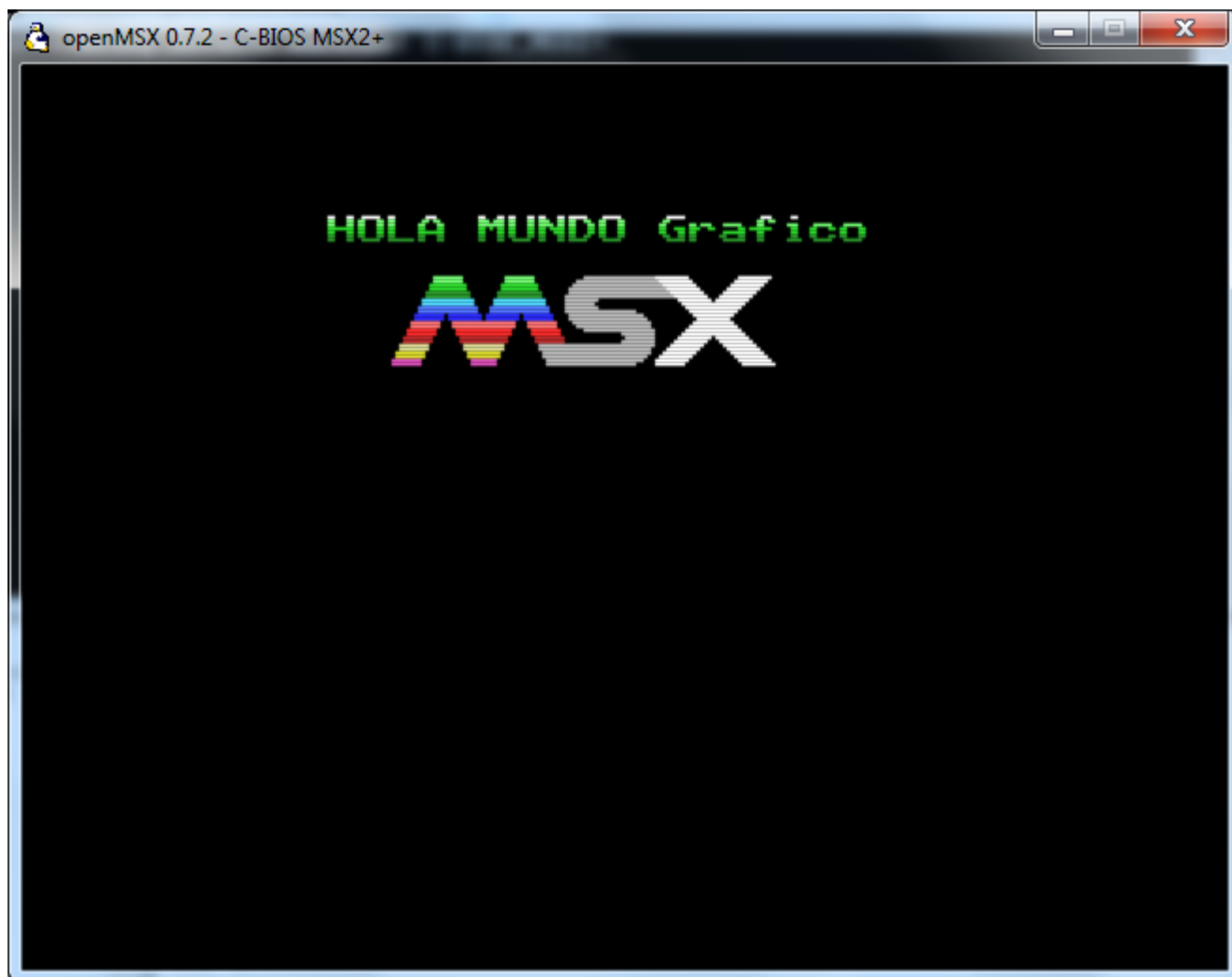
Escritor y Filósofo.

(Bilbao 1864 - Salamanca 1936)

TUTORIAL DE INICIACION A LA PROGRAMACION EN LENGUAJE ENSAMBLADOR PARA MSX

4ª PARTE – EL MUNDO DE LOS GRAFICOS 2

En esta parte del tutorial vamos a retomar el código del [Hola Mundo Grafico](#) de la tercera entrega y añadirle un colorido especial a nuestro set de caracteres, además incorporaremos un grafico o logotipo y veremos cómo trabajarlo desde el código, así como mostrarlo en la pantalla, volveremos a repasar procesos dados en la tercera entrega, hasta conseguir la imagen que vemos debajo de este texto.

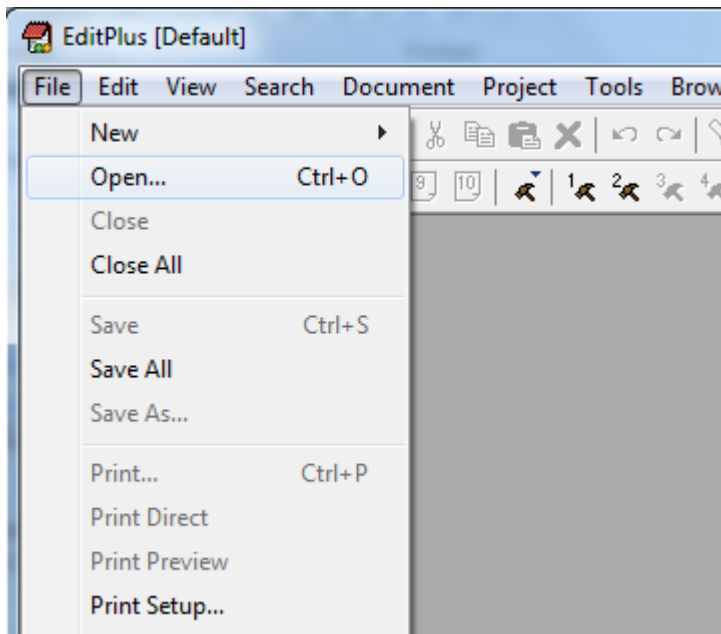


Este será el resultado final de nuestro nuevo tutorial, pero antes de llegar aquí veremos una opción intermedia. A esta versión intermedia la llamaré "HolaMundoGrafico2.asm" que tenéis incluida en los ficheros suministrados para este tutorial aquí. [Tutorial4.rar](#)
<http://www.dimensionzgames.com/wp-content/uploads/downloads/2012/02/Tutorial4.rar>

Quiero comentar que el grafico del Logo del MSX lo he creado y dibujado en GIMP y después en la segunda parte veremos cómo importar nuestros gráficos al programa nMSXtiles.
El colorido que le he impreso al logo del MSX es una referencia a la paleta original del MSX empleando los 16 colores originales de nuestro querido estándar.

Todo lo que aprendáis en esta entrega os servirá para crearos vuestros menús o pantallas de inicio de tus ROM's, o si vuestro objetivo es crear un videojuego las pantallas de introducción.

Vamos con el Código del [HolaMundoGrafico2.asm](#)



Ejecuta el [EditPlus 3](#) desde el menú inicio

Pulsa en los menús en...

[File – Open](#)

Abre el fichero [HolaMundoGrafico1.asm](#) de la tercera entrega del tutorial.

Cuando lo tengas abierto antes de modificar este fichero realiza lo siguiente

Pulsa en los menús en [File – Save As...](#) y lo guardas en el directorio que quieras y con el nombre que quieras. Yo le pongo [HolaMundoGrafico2.asm](#)

Este es el código del HolaMundoGrafico1 que ahora hemos grabado como HolaMundoGrafico2.asm

```
-----  
; Nombre de nuestro programa  
; Hola Mundo Grafico - 24/09/2011  
; Versión 1  
-----
```

Lo primero es cambiar en el código el nombre de nuestro proyecto, modificáis las líneas de la fecha de creación y el número de versión.

```
-----  
; Nombre de nuestro programa  
; Hola Mundo Grafico - 28/09/2011  
; Versión 2  
-----
```

Lo segundo que vamos a modificar ya que queremos dar un color especial a todo nuestro set de letras es este apartado que te pongo debajo de este texto y que te explico en la tercera entrega.

```
-----  
; Colocar el color de las letras en VRAM en la CLRTBL  
ld hl, CLRTBL+(32*8) ; Empezar en el CHR 32 de la CLRTBL  
ld bc, (32*24) ; numero de caracteres  
ld a, 0Fh ; Valor a rellenar  
call FILVRM ; BIOS -Fill block of VRAM with data byte  
-----
```

La rutina **FILVRM** está situada en la BIOS y tiene unos parámetros de entrada, que son los que te describo a continuación:

El registro **HL** tiene que apuntar a la dirección de memoria de la VRAM donde queremos empezar a colocar los bytes

En el ejemplo que nos ocupa queremos empezar a colocar bytes en el carácter 32 de la [Colour Table – CLRTBL](#)

```
ld hl, CLRTBL+(32*8) ; Empezar en el CHR 32 de la CLRTBL
```

En el registro **BC** le decimos cuantos son los bytes que hay que rellenar.

El set de caracteres ocupa 32 caracteres de ancho por 3 caracteres de alto, pero recuerda que cada CHR tiene de alto 8 bytes, entonces hay que multiplicar 3 x 8=24 bytes de altura, por eso en el registro **BC** pongo 32 x 24

```
ld bc, (32*24) ; numero de caracteres
```

En el registro **A** le decimos el byte que tiene que usar para rellenar toda la zona seleccionada.

En el ejemplo que nos ocupa el Color de la tinta es blanco F y el fondo negro 0, o lo que es lo mismo **0Fh** en hexa. 16 Colores de fondo y 16 colores de tinta, recuerda que los colores empiezan con el 0 y terminan en el 15 si mira la paleta del MSX veras que el blanco es el 15 y el negro el 0 - 15 en hexadecimal es F y el negro 0, resultado 0Fh si quieres cambiar colores ya sabes cómo tienes que hacerlo.

```
ld a, 0Fh ; Valor a rellenar
```

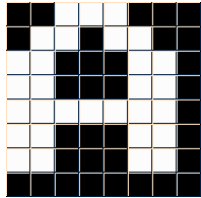


Paleta MSX

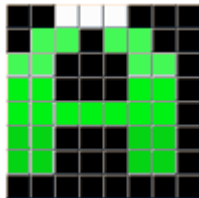
Esta es la llamada a la rutina que empezara escribiendo el byte 0Fh en VRAM empezando en CLRTBL+256 y no parara hasta que se rellenen 32x24=768 Bytes que son las 3 filas de 32 CHR que ocupa el set de caracteres en la [Colour Table - CLRTBL](#)

```
call FILVRM ; BIOS -Fill block of VRAM with data byte  
-----
```

Haciendo esto rellenábamos los 96 CHRs cada uno con sus 8 bytes en la **Colour Table - CLRTBL** para dar color blanco fondo negro a todas las letras de nuestro set de caracteres.



Byte 0 - 0Fh
Byte 1 - 0Fh
Byte 2 - 0Fh
Byte 3 - 0Fh
Byte 4 - 0Fh
Byte 5 - 0Fh
Byte 6 - 0Fh
Byte 7 - 0Fh



Byte 0 - 0Fh
Byte 1 - 03h
Byte 2 - 03h
Byte 3 - 02h
Byte 4 - 02h
Byte 5 - 0Ch
Byte 6 - 0Ch
Byte 7 - 0Ch

Este es el nuevo color que vamos a darle a nuestro set de caracteres.

Puedes probar combinaciones de colores y darle tu mismo el colorido que quieras, pero deberás modificar la en el código los valores en **TBL_MICLR** y poner los valores que queráis utilizar.

Ahora vamos con lo que hay que cambiar o quitar dentro del código en ensamblador.

Esta es la parte que tenéis que quitar o borrar en el código, o bien modificarlo con el código de abajo.

```
; Colocar el color de las letras en VRAM en la CLRTBL
ld    hl, CLRTBL+(32*8)      ; Empezar en el CHR 32 de la CLRTBL
ld    bc, 32*24              ; numero de caracteres
ld    a, 0Fh                 ; Valor a rellenar
call  FILVRM                 ; BIOS -Fill block of VRAM with data byte
```

En el mismo lugar del código que has borrado tenéis que colocar este otro fragmento de código.

```
; Colocar el Multi-color de las letras en VRAM en la CLRTBL
ld    hl, TBL_MICLR          ; Tabla con el CHR de Color
ld    de, CLRTBL+(32*8)      ; Empezar en el CHR 32 de la CLRTBL
ld    b, (32*3)              ; Numero de caracteres
call  COPY_BLOCK             ; Rutina Encargada de realizar el proceso
```

Seguro que ya adivináis que es lo que va a realizar esta parte del código.

Cargamos **HL** apuntando a la tabla de 8 bytes del Color que daremos a los CHRs

Cargamos **DE** apuntando a la dirección **CLRTBL+(32*8)** en VRAM para que empiece en el CHR nº32

Cargamos el registro **B** con el número de CHRs que hay que rellenar con la tabla del color.

Y con esos parámetros de entrada llamamos a una nueva rutina que vamos a crear en nuestro código llamada **COPY_BLOCK** que será la encargada de tomar los 8 bytes de color que componen la tabla, y que vamos a aplicar al set de caracteres en la **Colour Table - CLRTBL** en VRAM.

A estas alturas solo voy a utilizar los nombres cortos en las descripciones de las zonas en VRAM.

Justo debajo en el apartado del **TXT_HOLA** añadiremos la tabla del Multi-color que queremos dar a nuestro set de caracteres añadiendo este código.

```
; Cadena de Texto a visualizar en la pantalla
TXT_HOLA:
db    "HOLA MUNDO Grafico"
```

```
; Tabla con el Multi-color para los CHRs
```

```
TBL_MICLR:
db    0Fh, 03h, 03h, 02h, 02h, 0Ch, 0Ch, 0Ch ; este es el color que hemos visto arriba en los bytes de la letra A
```

Esta es la rutina [COPY_BLOCK](#) que tienes que añadir justo debajo de lo que acabas de añadir.

```
-----  
; Copia de manera secuencial bloques de 8 bytes a la VRAM  
; Parámetros: HL = Dirección de Origen en RAM-ROM  
;             DE = Dirección de Destino en VRAM  
;             B  = Numero de CHRs a copiar máximo 256 CHRs  
-----  
COPY_BLOCK:  
    push    bc           ; Guardamos cuantos CHRs rellenaremos  
    push    hl           ; Guardamos la dirección de origen  
    push    de           ; guardamos la dirección de destino  
    ld      bc,8         ; copiamos los primeros 8 bytes  
    call    LDIRVM       ; BIOS - Copy block to VRAM, from memory  
    pop     hl           ; Recuperamos la dirección de destino  
    ld      bc,8         ; 8 bytes que sumaremos a la dirección  
    add     hl,bc        ; el destino en VRAM será 8 bytes mas  
    ex      de,hl        ; pasamos el registro HL al registro DE  
    pop     hl           ; recuperamos la dirección de origen  
    pop     bc           ; recuperamos el numero de CHRs que quedan  
    djnz    COPY_BLOCK   ; restamos 1 al número de CHRs y repetimos..  
    ; ...todo el proceso hasta llegar a cero CHRs  
    ret                ; salimos cuando esté finalizado  
-----
```

Como puedes observar en la rutina que está muy comentadita, se llama a la rutina en [BIOS LDIRVM](#) encargada de llevar bytes de ROM-RAM a VRAM en este caso va copiando de 8 en 8 bytes la tabla del color a cada carácter en la [CLRTBL](#) finalizando el bucle cuando el numero de CHRs sea cero.

La [CLRTBL](#) después de llamar a [COPY_BLOCK](#) quedará como ves en la imagen. Banco 0



Ya puedes grabar el fichero compilar y ejecutar nuestro código [HolaMundoGrafico2.asm](#)



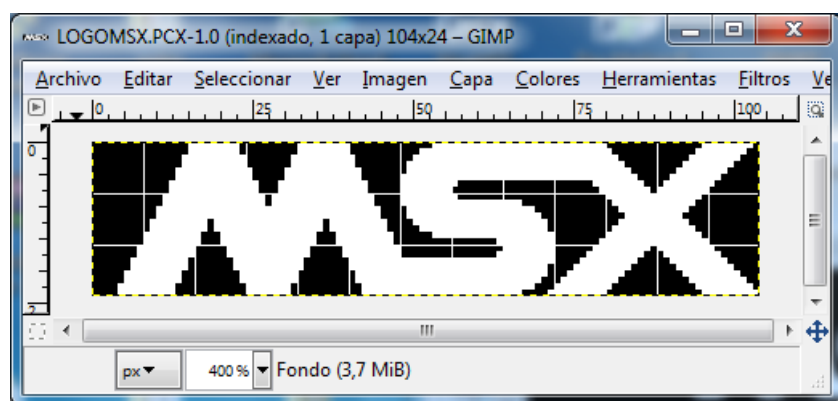
Y voila, a partir de ahora nuestros textos aparecen ya con el Multi-color. Elegido

Veis como era muy fácil de implementar el color.

Vamos ahora a explicar la parte que mostrara el logotipo o grafico junto al Hola Mundo. Lo primero será crear el logotipo o grafico que vamos a insertar en la pantalla. Así que abre el GIMP y manos a la obra, crea un grafico como te enseñe en la 3ª entrega cuando realizamos el set de caracteres. Os pongo unos ScreenCaps del proceso de creación. El fichero se llama [LOGOMSX2.pcx](#)



Primero dibujo al pixel, ampliado al 1600% sobre el tamaño original, creando el cuerpo del dibujo.



Después con la herramienta de relleno realizo un relleno interior en color blanco.

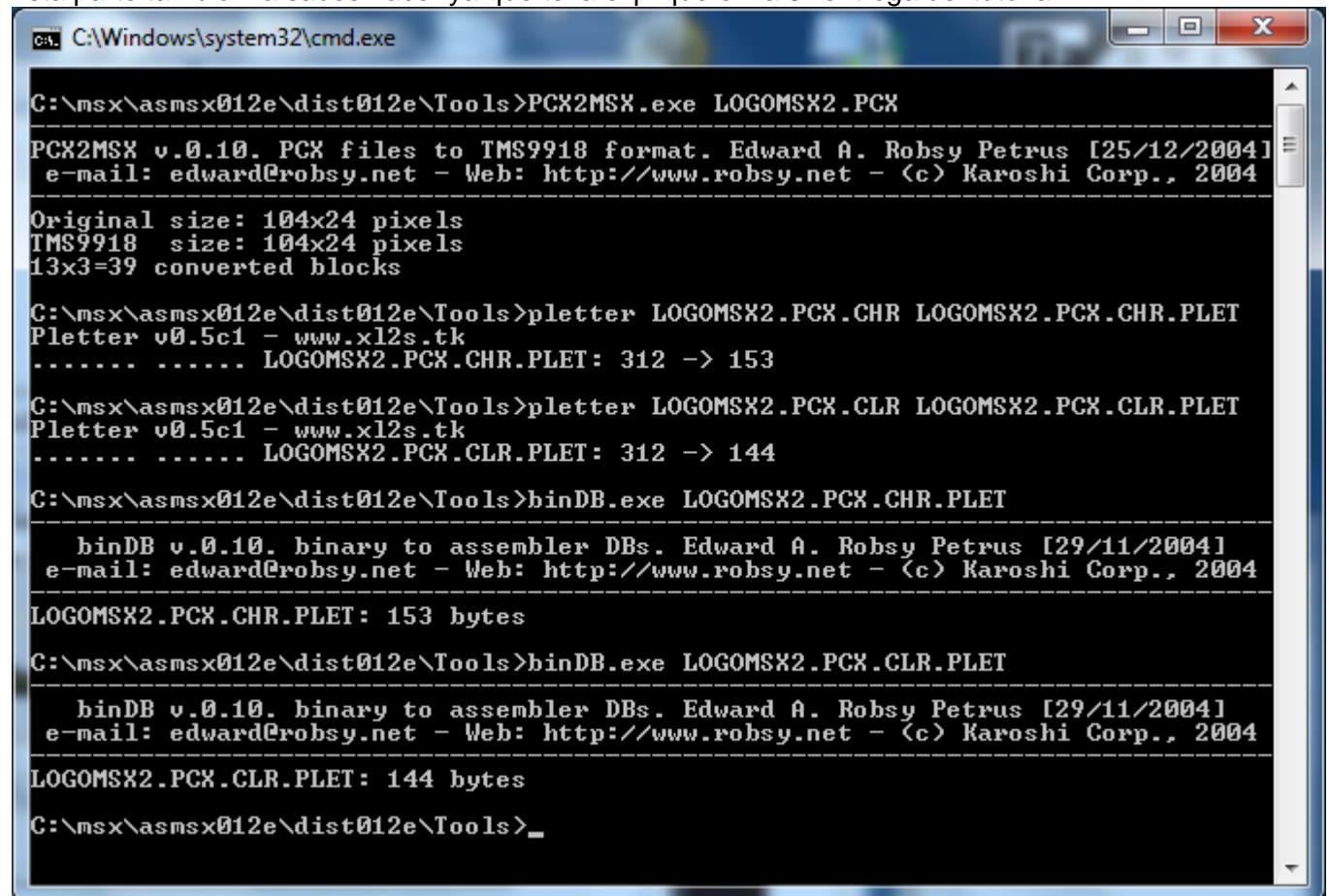


Finalmente le doy el color al logotipo con la precaución de no utilizar más de dos colores por CHR con la ayuda del Grid 8x1 en Gimp. El PCX2MSX te avisará del Error si empleas más de 2 colores por byte.

Ojo con la paleta de colores ya que cada vez que creamos una nueva imagen GIMP toma la paleta por defecto, en el menú [Colores – Mapa – Establecer el mapa de colores](#) pulsas en Default y selecciona de la lista de paletas la mía [VDP-MSX-TMS9918-\(Paleta PEPE\)](#) pulsa el botón aceptar y listo. Esto te lo explico si no te quieres llevar sorpresas después, cuando veas el color en la pantalla del MSX.

Guardas el fichero de imagen en formato PCX con el nombre de [LOGOMSX2.PCX](#) en el directorio donde tienes las herramientas, pletter.exe, pcx2msx.exe y el bindb.exe.

Esta parte también la sabes hacer ya que te la explique en la 3ª entrega del tutorial.



```
C:\Windows\system32\cmd.exe

C:\msx\asmsx012e\dist012e\Tools>PCX2MSX.exe LOGOMSX2.PCX

PCX2MSX v.0.10. PCX files to TMS9918 format. Edward A. Robsy Petrus [25/12/2004]
e-mail: edward@robsy.net - Web: http://www.robsy.net - (c) Karoshi Corp., 2004

-----
Original size: 104x24 pixels
TMS9918 size: 104x24 pixels
13x3=39 converted blocks

C:\msx\asmsx012e\dist012e\Tools>pletter LOGOMSX2.PCX.CHR LOGOMSX2.PCX.CHR.PLET
Pletter v0.5c1 - www.x12s.tk
..... LOGOMSX2.PCX.CHR.PLET: 312 -> 153

C:\msx\asmsx012e\dist012e\Tools>pletter LOGOMSX2.PCX.CLR LOGOMSX2.PCX.CLR.PLET
Pletter v0.5c1 - www.x12s.tk
..... LOGOMSX2.PCX.CLR.PLET: 312 -> 144

C:\msx\asmsx012e\dist012e\Tools>binDB.exe LOGOMSX2.PCX.CHR.PLET

binDB v.0.10. binary to assembler DBs. Edward A. Robsy Petrus [29/11/2004]
e-mail: edward@robsy.net - Web: http://www.robsy.net - (c) Karoshi Corp., 2004

LOGOMSX2.PCX.CHR.PLET: 153 bytes

C:\msx\asmsx012e\dist012e\Tools>binDB.exe LOGOMSX2.PCX.CLR.PLET

binDB v.0.10. binary to assembler DBs. Edward A. Robsy Petrus [29/11/2004]
e-mail: edward@robsy.net - Web: http://www.robsy.net - (c) Karoshi Corp., 2004

LOGOMSX2.PCX.CLR.PLET: 144 bytes

C:\msx\asmsx012e\dist012e\Tools>_
```

Te muestro esta captura para que veas todo el proceso.

- 1 - Generamos los ficheros binarios separando los **CHRs** y los **CLRs** con el PCX2MSX.exe
- 2 - Comprimimos el fichero binario **LOGOMSX2.PCX.CHR** con pletter que son los caracteres.
- 3 - Comprimimos el fichero binario **LOGOMSX2.PCX.CLR** con pletter que son los colores de los CHRs.
- 4 - Convertimos el Binario comprimido de los **CHRs** a formato ensamblador con el binDB.exe
- 5 - Convertimos el Binario comprimido de los **CLRs** a formato ensamblador con el binDB.exe

Ya tenemos creado nuestros ficheros **.asm** con los bytes comprimidos de nuestro logo.

El Primero con los caracteres **LOGOMSX2.PCX.CHR.PLET.asm** al que le llamaremos **LOGO_CHR**:

```
; LOGOMSX2.PCX.CHR.PLET - binary dump
; generated by binDB v.0.10
LOGO_CHR:
    db 01Eh,0FFh,000h,000h,0FEh,0FEh,0FCh,0FCh
    db 0F8h,0F8h,0F0h,01Ah,0F0h,001h,001h,0C1h
```

El Segundo con los colores **LOGOMSX2.PCX.CLR.PLET.asm** al que le llamaremos **LOGO_CLR**:

```
; LOGOMSX2.PCX.CLR.PLET - binary dump
; generated by binDB v.0.10
LOGO_CLR:
    db 01Eh,000h,000h,000h,003h,003h,002h,002h
    db 00Ch,00Ch,007h,040h,007h,007h,022h,022h
```

Estas etiquetas se las ponemos al principio de cada binario para después por código localizar estos datos, y que deberéis añadir al código del **HolaMundoGrafico3.asm** que vamos a crear.

Repetimos el proceso descrito al principio del tutorial abre el [HolaMundoGrafico2.asm](#) y guarda el código como [HolaMundoGrafico3.asm](#) y modifica la versión y la fecha, ya que vamos a modificar este código para añadir el logo del MSX.

Vamos con lo nuevo que hay que añadir al código:

Justo debajo de colocar la cadena de texto agregaremos los nuevos fragmentos de código.

```
; Colocar la cadena de texto en la pantalla
; LOCATE 6,2: PRINT "Hola Mundo Grafico"
    ld    hl,TXT_HOLA          ; Dirección donde tenemos el texto
    ld    de,NAMBTBL+6+(2*32)  ; LOCATE 6,2 : Destino la NAMBTBL en VRAM
    ld    bc,18                ; Numero de CHR que tiene el texto
    call  LDIRVM               ; BIOS - Copy block to VRAM, from memory
```

Coloca estos dos grupos de código en este punto.

```
; Colocar el logo del msx
    ld    hl,LOGO_CHR          ; Dirección de los CHR del logo MSX
    ld    de,CHRTBL+(128*8)    ; los situaremos a partir del CHR nº128 en la CHRTBL
    call  DEPLET               ; Descomprimir en VRAM

; Colocar el color del logo del msx
    ld    hl,LOGO_CLR          ; Dirección de los CLR del logo MSX
    ld    de,CLRTBL+(128*8)    ; los situaremos a partir del CHR nº128 en la CHRTBL
    call  DEPLET               ; Descomprimir en VRAM
```

Las explicaciones de esta parte son muy similares a lo explicado en la 3ª entrega del tutorial, donde descomprimos el set de caracteres.

Básicamente lo que realiza este apartado es descomprimir los bytes de los CHRs del logo colocándolos en la CHRTBL empezando en el primer CHR de la 5ª Fila o lo que es lo mismo en el [CHR nº128](#)

El segundo apartado descomprime los bytes de los colores de los CHRs y los colocamos en la CLRTBL empezando en el primer CHR de la 5ª Fila o lo que es lo mismo en el [CHR nº128](#).

Os pongo una imagen con el resultado final la CHRTBL y la CLRTBL, el 1º tercio o banco quedaría así.



Este apartado te lo explico por si quieres modificar los tutoriales o colocar texto o gráficos en cualquier zona de la pantalla para vuestras futuras creaciones.

Recuerdas que en la 3ª entrega os comente que la VRAM está dividida en tres bancos de 256 CHRs cada uno, solo podemos colocar datos en el primer tercio de la pantalla, ya que solo hemos colocado CHRs en el primer banco, si queremos mostrar textos o gráficos en toda la pantalla debemos colocar o repetir los mismos caracteres en los 3 tercios o bancos de la CHRTBL y CLRTBL.

No es muy difícil y con lo que has aprendido tu mismo ya lo puedes hacer te doy una pista.

256 CHRs tiene un banco multiplicado por 8 bytes que tiene cada CHR nos da un total de 2048 Bytes.

```
; Colocar los gráficos de las letras en VRAM en la CHRTBL en el tercio 2
    ld    hl,SET2_PLET         ; set de CHR de las letras
    ld    de,CHRTBL+2048+(32*8) ; Empezar en el CHR 32 del banco 1
    call  DEPLET               ; Descomprimir en VRAM

; Colocar los gráficos de las letras en VRAM en la CHRTBL en el tercio 3
    ld    hl,SET2_PLET         ; set de CHR de las letras
    ld    de,CHRTBL+4096+(32*8) ; Empezar en el CHR 32 del banco 2
    call  DEPLET               ; Descomprimir en VRAM
```




Haciendo esto ya podrías poner texto o el logo del MSX en cualquier Fila y Columna de la pantalla. Seguro que tu mismo podrás crear el HolaMundoGrafico4.asm si quieres probar cosas.

Sigamos con el HolaMundoGrafico3 ya tenemos los CHR's y los CLR's en la VRAM ahora nos queda colocar en la NAMTBL los números de los CHR's que pertenecen al logo del MSX para que este se muestre en la pantalla, casi de la misma manera como lo hacemos con el texto.

El logo del MSX está compuesto por 13 CHR's de ancho por 3 líneas de CHR's de alto y como hemos colocado el primer CHR del logo en la CHRTBL en el numero 128 haremos lo siguiente.

```
Locate 8,4; Print chr$(128);chr$(129);chr$(130);chr$(131);chr$(132);chr$(133);chr$(134);chr$(135);
;chr$(136);chr$(137);chr$(138);chr$(139);chr$(140)

; Colocar los 13 primeros CHR's del logo
ld hl,NAMLOGO ; Primera fila del mapeado
ld de,NAMTBL+8+(4*32) ; LOCATE 8,4
ld bc,13 ; Numero de CHR's
call LDIRVM ; BIOS - Copy block to VRAM, from memory

Locate 8,5; Print chr$(141);chr$(142);chr$(143);chr$(144);chr$(145);chr$(146);chr$(147);chr$(148);
;chr$(149);chr$(150);chr$(151);chr$(152);chr$(153)

; Colocar los 13 segundos CHR's del logo
ld hl,NAMLOGO+13 ; Segunda fila del mapeado
ld de,NAMTBL+8+(5*32) ; LOCATE 8,5
ld bc,13 ; Numero de CHR's
call LDIRVM ; BIOS - Copy block to VRAM, from memory

Locate 8,6; Print chr$(154);chr$(155);chr$(156);chr$(157);chr$(158);chr$(159);chr$(160);chr$(161);
;chr$(162);chr$(163);chr$(164);chr$(165);chr$(166)

; Colocar los 13 terceros CHR's del logo
ld hl,NAMLOGO+13+13 ; Tercera fila del mapeado
ld de,NAMTBL+8+(6*32) ; LOCATE 8,6
ld bc,13 ; Numero de CHR's
call LDIRVM ; BIOS - Copy block to VRAM, from memory
```

Por eso he creado una tabla llamada NAMLOGO que toma los valores de los bytes que colocaremos en la NAMTBL justo debajo de los bytes comprimidos y del color del Logo del MSX. ([ver mas abajo](#))

Al final de los bytes comprimidos de set de caracteres es donde debes agregar los bytes comprimidos del Logo del MSX que previamente os he explicado.

```
-----
; LOGOMSX2.PCX.CHR.PLET - binary dump
; generated by binDB v.0.10
;
LOGO_CHR:
db 01Eh,0FFh,000h,000h,0FEh,0FEh,0FCh,0FCh
db 0F8h,0F8h,0F0h,01Ah,0F0h,001h,001h,0C1h
db 011h,07Ch,07Ch,038h,038h,0F4h,00Dh,000h
db 07Fh,000h,07Fh,03Fh,03Fh,01Fh,01Fh,0F8h
db 0E0h,0C0h,035h,080h,080h,0B5h,013h,080h
db 01Eh,080h,0C0h,0E0h,0F0h,0F8h,0FCh,050h
db 0FEh,012h,024h,00Fh,007h,003h,055h,001h
db 04Eh,04Ch,02Ah,001h,029h,001h,003h,007h
db 00Fh,01Fh,03Fh,0ACh,027h,0E0h,0B5h,014h
db 000h,040h,020h,020h,070h,070h,010h,010h
db 0D6h,042h,005h,008h,008h,01Ch,01Ch,00Fh
db 03Ch,03Ah,03Dh,003h,085h,025h,09Fh,051h
db 0F8h,04Ch,061h,057h,01Fh,054h,0FBh,09Bh
db 0AEh,00Fh,01Dh,05Eh,0EEh,05Fh,06Eh,07Fh
db 0FFh,055h,0C7h,019h,02Dh,047h,055h,04Bh
db 00Bh,00Fh,013h,0F4h,01Dh,013h,03Eh,03Eh
db 03Eh,07Fh,08Ch,0E6h,061h,0C0h,0E6h,051h
db 0B4h,024h,037h,01Eh,0F3h,0BEh,05Bh,0BFh
db 0CFh,0DFh,01Fh,0DEh,0FFh,0FFh,0FFh,0FFh
db 0F8h
```

```
; .INCBIN "LOGOMSX2.PCX.CHR.PLET"
;-----
```

```
-----
; LOGOMSX2.PCX.CLR.PLET - binary dump
; generated by binDB v.0.10
;
LOGO_CLR:
db 01Eh,000h,000h,000h,003h,003h,002h,002h
db 00Ch,00Ch,007h,040h,007h,007h,022h,022h
db 0CCh,0CCh,077h,05Bh,077h,011h,01Eh,00Fh
db 033h,033h,0CEh,00Fh,00Eh,000h,07Ch,0EEh
db 000h,00Eh,06Dh,000h,007h,01Ch,01Dh,0FFh
db 0EFh,000h,00Fh,01Eh,00Fh,000h,00Fh,08Fh
db 000h,0FFh,01Fh,007h,000h,000h,000h,005h
db 005h,004h,004h,009h,009h,088h,004h,088h
db 055h,055h,044h,044h,007h,008h,051h,008h
db 00Fh,007h,099h,099h,0BDh,00Fh,0D5h,01Fh
db 007h,05Bh,0D3h,069h,05Fh,00Eh,0EFh,068h
db 03Ch,078h,046h,0F0h,058h,0FFh,0FFh,0FBh
db 067h,0E0h,05Fh,006h,006h,00Bh,00Bh,003h
db 00Ah,00Ah,00Dh,00Dh,066h,066h,06Dh,007h
db 0A8h,00Fh,017h,00Fh,0BBh,0BBh,036h,0AAh
db 0AAh,00Fh,0EBh,027h,00Fh,067h,061h,000h
db 0DBh,0C1h,09Ch,0D4h,000h,00Fh,0F6h,05Fh
db 0BFh,0DFh,03Fh,0D7h,0FFh,0FFh,0FFh,0F0h
```

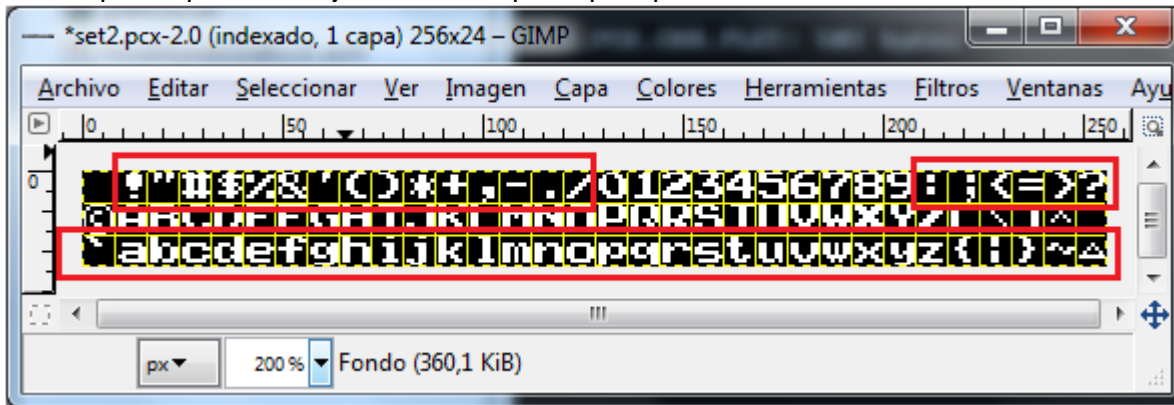
```
; .INCBIN "LOGOMSX2.PCX.CLR.PLET"
;-----
```

```
-----
; tabla con los CHRs del logoMSX
NAMLOGO:
db 128,129,130,131,132,133,134,135,136,137,138,139,140
db 141,142,143,144,145,146,147,148,149,150,151,152,153
db 154,155,156,157,158,159,160,161,162,163,164,165,166
;-----
```

```
-----
; FINAL DE NUESTRO CODIGO EN ENSAMBLADOR.
;-----
```

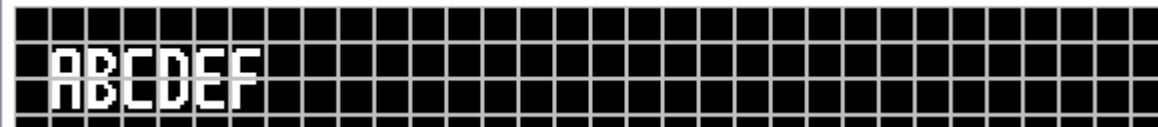
Ahora ya puedes grabar, compilar y ejecutar el Código del HolaMundo3.asm el resultado final ya lo conoces es la primera imagen de este manual.

Cosas que se pueden mejorar o trucos para que aprendáis.



Las letras en minúsculas no se suelen utilizar, puedes utilizar esos 32 CHRs para colocar más gráficos, o incluso para colocar el LOGO del MSX en esos CHRs al igual que con los símbolos otros 21 CHRs salvo que quieras usarlos. El resto de CHRs debe ir en su sitio. (Por la Norma del ASCII)
Yo añadido en uno de los CHRs no usados en el ultimo, nuestra querida letra Ñ para usarla.

Podrías crearte un set de caracteres de la siguiente manera para tener unas letras de doble tamaño.



En las Mayúsculas creas la parte de arriba de las letras y en las Minúsculas creas la parte de abajo. A la hora de crear el menú o colocar los CHRs en la NAMTBL en el código harías lo siguiente.

```
; Cadena de Texto a visualizar en la pantalla
TXT_MA:
    db      "ABCDEF"
TXT_MI:
    db      "abcdef"

; Colocar la cadena de texto en la pantalla
; LOCATE 6,2: PRINT "ABCDEF"
    ld      hl,TXT_MA          ; Dirección donde tenemos el texto
    ld      de,NAMTBL+6+(2*32) ; LOCATE 6,2 : Destino la NAMTBL en VRAM
    ld      bc,6               ; Numero de CHRs que tiene el texto
    call    LDIRVM             ; BIOS - Copy block to VRAM, from memory

; LOCATE 6,3: PRINT "abcdef"
    ld      hl,TXT_MI          ; Dirección donde tenemos el texto
    ld      de,NAMTBL+6+(3*32) ; LOCATE 6,3 : Destino la NAMTBL en VRAM
    ld      bc,6               ; Numero de CHRs que tiene el texto
    call    LDIRVM             ; BIOS - Copy block to VRAM, from memory
```

El resultado en pantalla sería el de la imagen que te pongo aquí debajo con Multi-color o sin el.



Os dejo a vosotros que experimentéis con esto... a ver si veo vuestros resultados en los Foros.

Espero que haya sido de vuestro total agrado y nos vemos en el próximo tutorial para finalizar con el mundo de los gráficos. Donde veremos otra forma de crear menús con texto y gráficos, importar imágenes al nMSXtiles y trabajar con este.

José Vila Cuadrillero

"ES DETESTABLE ESA AVARICIA ESPIRITUAL QUE TIENEN, LOS QUE SABIENDO ALGO, NO PROCURAN LA TRANSMISION DE ESOS CONOCIMIENTOS."

Miguel de Unamuno

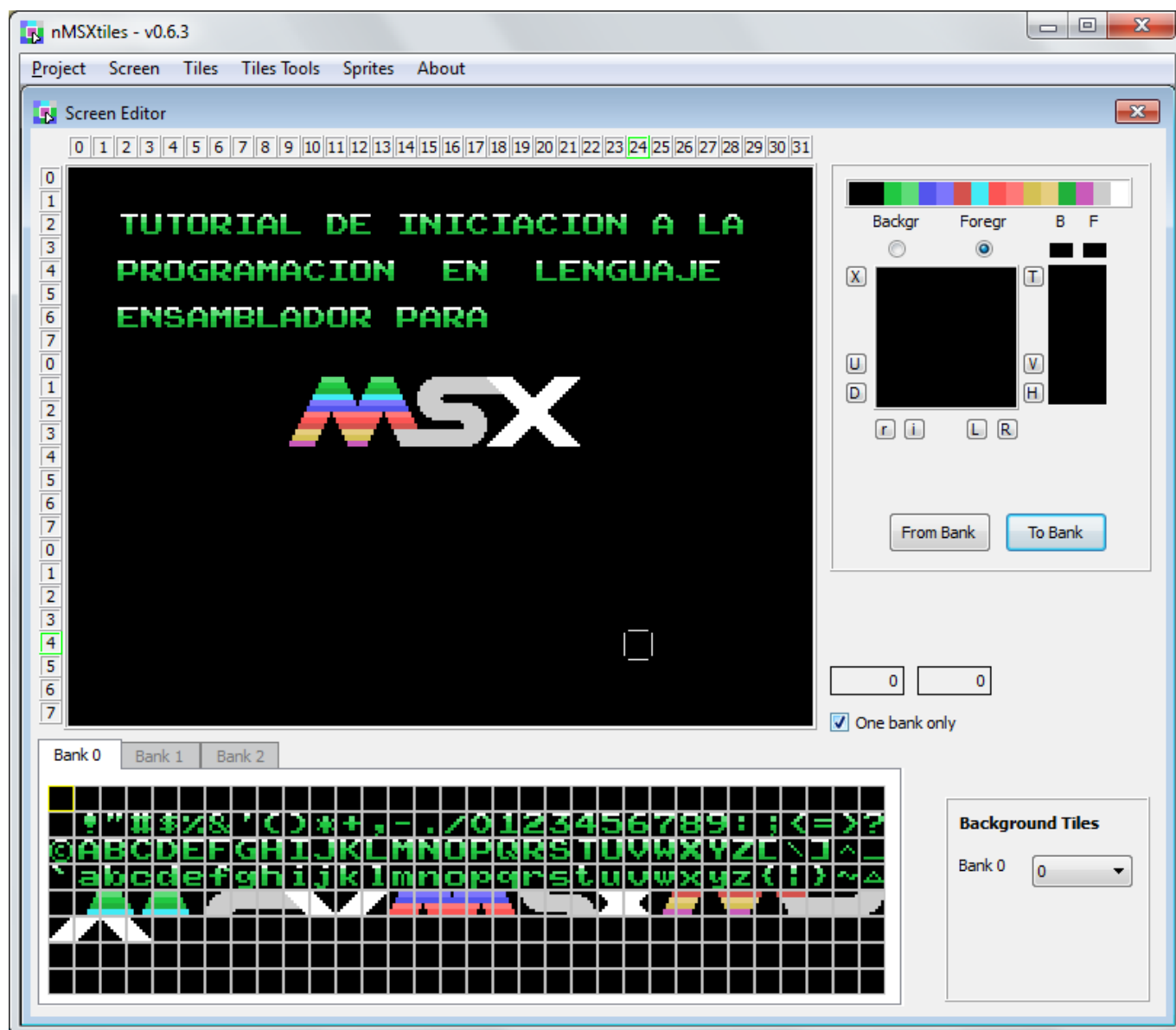
Escritor y Filósofo.

(Bilbao 1864 - Salamanca 1936)

TUTORIAL DE INICIACION A LA PROGRAMACION EN LENGUAJE ENSAMBLADOR PARA MSX

5ª PARTE – EL MUNDO DE LOS GRAFICOS y 3

En esta parte del tutorial vamos a cambiar la forma de trabajar para mostrar el hola mundo grafico de otra manera que emplea menos código y bytes, tengo que decir que esta es la forma que yo uso para crear los menús en mi juego JumpinG. Veremos cómo importar nuestras imágenes o gráficos al nMSXtiles de Ramon de las Heras, que ha tenido a bien modificarlo con unas opciones que le he pedido. Veremos cómo se trabaja con este programa que empleo para todo el tema de menús y mapeados. Y daremos por finalizada la entrega sobre el Mundo de los Gráficos.



Esta será la herramienta principal para esta entrega del tutorial, el texto y el logo que veis dentro del nMSXtiles será el resultado final de nuestro tutorial, al que le llamare “HolaMundoGrafico4.asm” y que tenéis incluido en los ficheros suministrados para este tutorial aquí. [Tutorial5.rar](#)

<http://www.dimensionzgames.com/wp-content/uploads/downloads/2012/02/Tutorial5.rar>

Lo primero que vamos a modificar es la paleta que usamos en GIMP, ya que el color transparente y el negro son el mismo y a la hora de importar nos puede hacer falta separar estos colores. Tienes que buscar dentro del directorio donde tienes instalado el GIMP hay una carpeta llamada [palettes](#) que es donde las almacena el GIMP o bien realizar una búsqueda en Windows del fichero de paleta que usamos [VDP-MSX-TMS9918-\(Paleta-PEPE\).gpl](#) puedes usar el EditPlus para modificarlo.

```

1 GIMP-Palette
2 Name: VDP-MSX-TMS9918-(Paleta-PEPE).gpl
3 Columns: 16
4 #
5 --0--0--0--0-- Transparent
6 --0--0--0--0-- black
7 -32-200--64-- medium-green
8 -94-220-120-- light-green
9 -84--85-237-- dark-blue
10 125-118-252-- light-blue
11 212--82--77-- dark-red
12 -66-235-245-- cyan
13 252--85--84-- medium-red
14 255-121-120-- light-red
15 212-193--84-- dark-yellow
16 230-206-128-- light-yellow
17 -33-176--59-- dark-green
18 201--91-186-- magenta
19 204-204-204-- gray
20 255-255-255-- Sin nombre
21

```

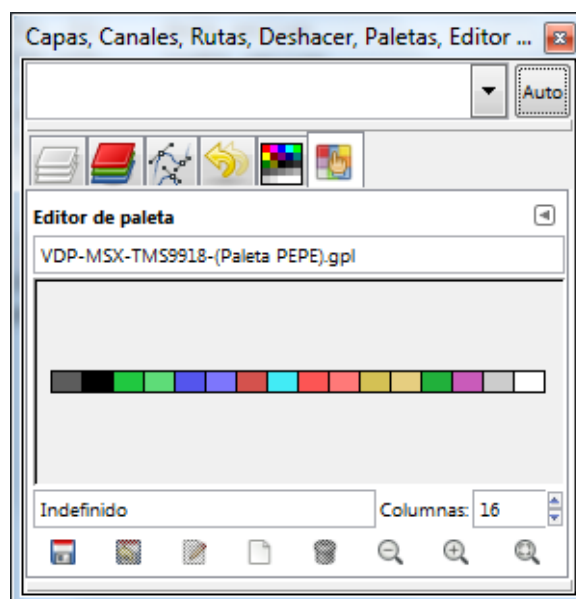
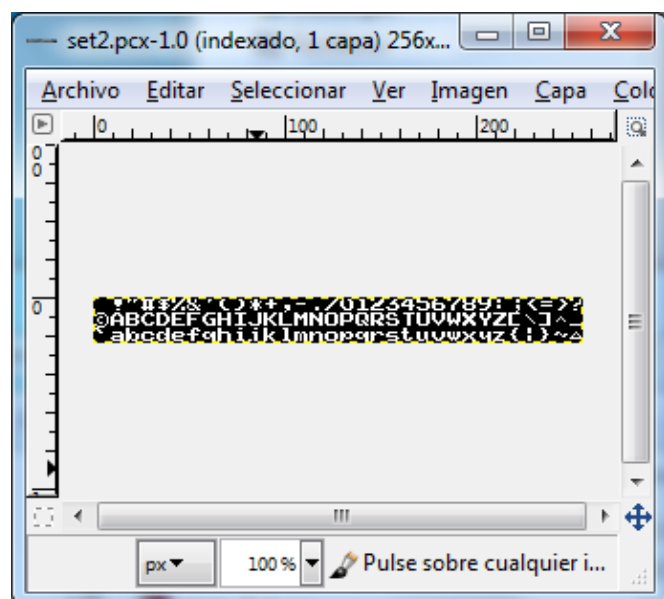
```

1 GIMP-Palette
2 Name: VDP-MSX-TMS9918-(Paleta-PEPE).gpl
3 Columns: 16
4 #
5 --92--92--92-- Transparent
6 --0--0--0--0-- black
7 -32-200--64-- medium-green
8 -94-220-120-- light-green
9 -84--85-237-- dark-blue
10 125-118-252-- light-blue
11 212--82--77-- dark-red
12 -66-235-245-- cyan
13 252--85--84-- medium-red
14 255-121-120-- light-red
15 212-193--84-- dark-yellow
16 230-206-128-- light-yellow
17 -33-176--59-- dark-green
18 201--91-186-- magenta
19 204-204-204-- gray
20 255-255-255-- Sin nombre
21

```

Aquí puedes ver que modificamos los tres 0 por tres 92 el color **Transparent**, de esta manera el negro será negro mientras que el transparente será gris oscuro. Guarda o graba el fichero.

Ahora vamos a abrir el set de caracteres con el GIMP fichero [set2.pcx](#)

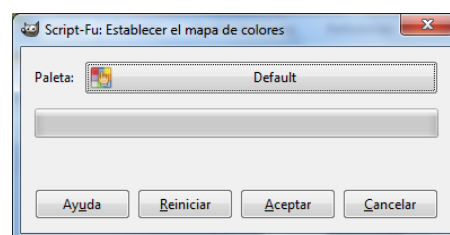


Aquí puedes ver la imagen PCX con nuestro set de caracteres y fíjate que en la paleta ahora tenemos el color Transparente en un gris oscuro.

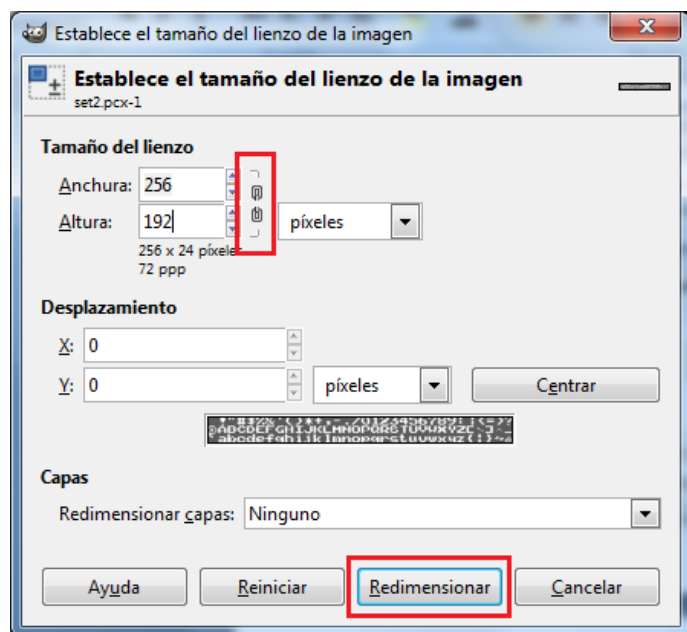
Lo primero siempre después de abrir una imagen es cambiar la paleta del GIMP en modo **DEFAULT** por la del MSX. Pula en los menús del GIMP en **COLORES – MAPA – Establecer el mapa de colores...** veras la imagen a la derecha de este texto.

Pula dentro de Default y en la ventana que se abre selecciona la paleta [VDP-MSX-TMS9918-\(Paleta-PEPE\)](#). Pula botón Cerrar.

Donde antes ponía **Default** ahora pone [VDP-MSX-TMS9918-\(Paleta-PEPE\)](#). Pula botón Aceptar.



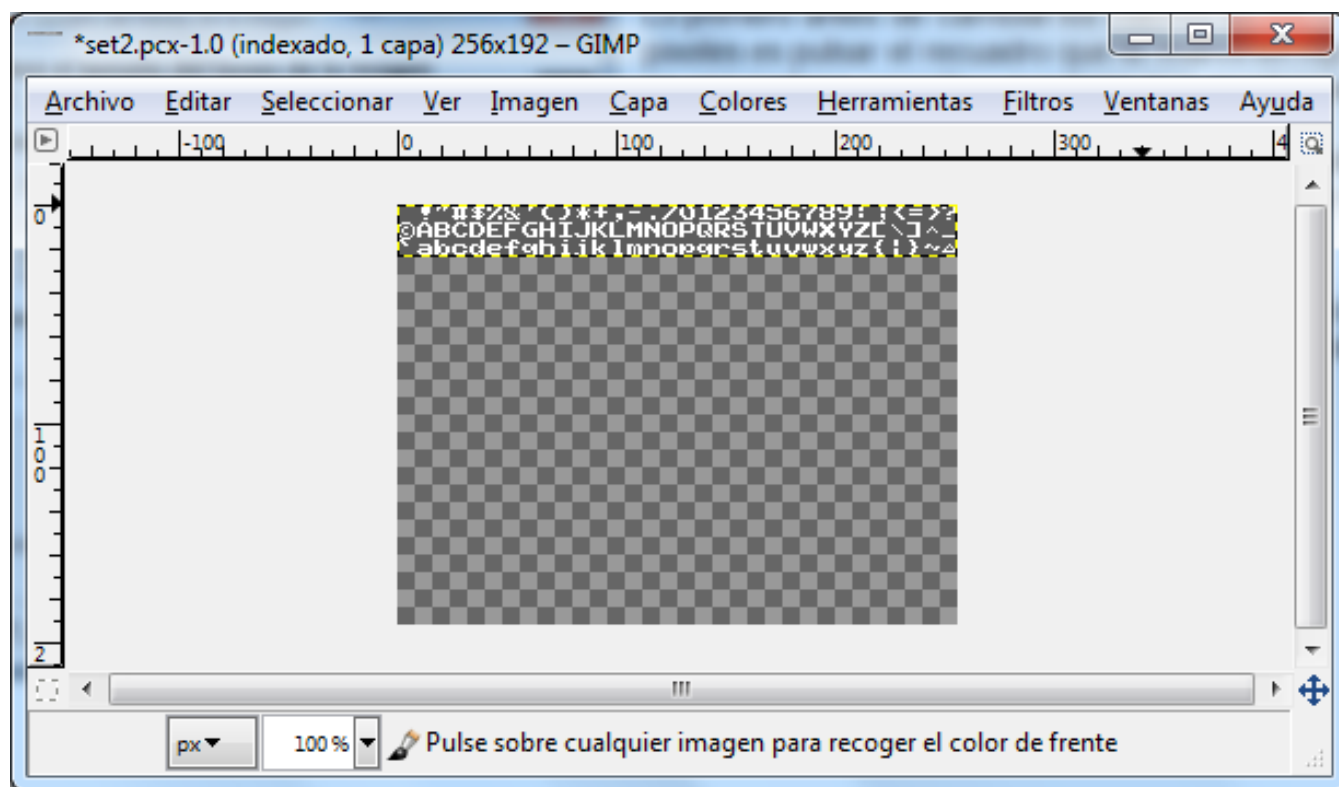
Ahora vamos a transformar el tamaño de la imagen a 256x192 ya que el importador del nMSXtiles no permite importar tamaños menores de esta resolución. Esto se realiza en los menús del GIMP en **IMAGEN-Tamaño del lienzo...** (**Esto no altera la imagen, solo cambia el tamaño.**)



Lo primero antes de cambiar los valores de los píxeles es pulsar el recuadro que te marco en rojo, es una cadena cerrada y que tiene que mostrarse como ves en esta imagen abierta, ya que esta opción es para que cree una imagen proporcional y nosotros lo que queremos es crear una imagen de un tamaño determinado.

Como nuestro ancho ya es 256 no lo cambiamos pero el alto que es 24 lo cambiamos a 192. Ahora pulsa en el botón Redimensionar. La ventana se cerrara, asegúrate que en **redimensionar capas** pone **ninguno**, sino se cierra esta ventana. Con esto ya tendremos nuestra imagen en **256x192 píxeles** que es la resolución del msx.

Fíjate que la imagen ha cambiado de tamaño pero sigue conservando intacto nuestro grafico.

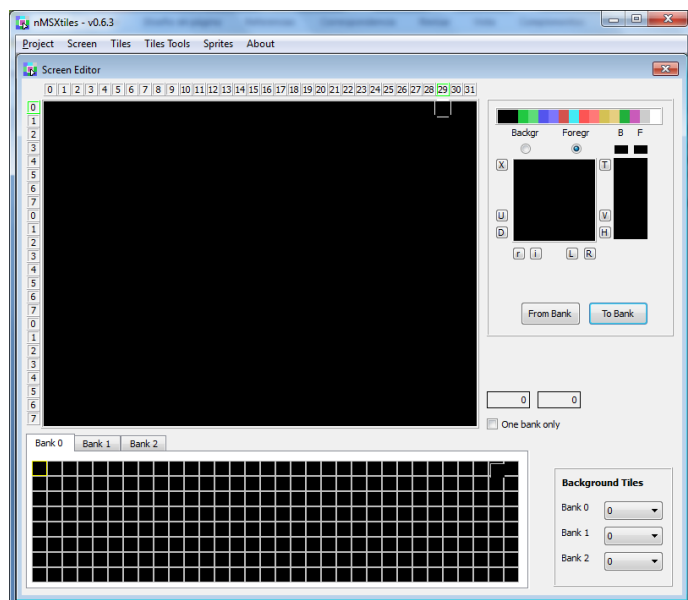


Ahora vamos a guardar la imagen en formato PNG para poderla importar con el nMSXtiles. Pulsa en el menú **Archivo** del GIMP en la opción **Guardar Como...**, en la ventana que se abre cambia el nombre **set2.pcx** a **set2.png** con esto le decimos que queremos grabarlo en PNG pulsa botón **Guardar**.

Veras que sale un mensaje de advertencia, ya que al cambiar el tamaño ha añadido una capa transparente y el PNG no puedes almacenar información de capas y otras cosas. Pulsa el botón **Exportar** y el solo se encargara de combinar las capas para grabar el PNG. En cuanto a los valores de exportación que salen en la otra ventana yo los dejo por defecto y pulso el botón **Guardar**

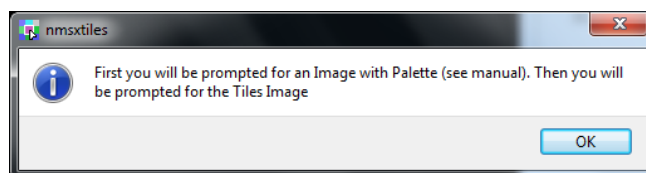
Ya estamos listos para importarlo al nMSXtiles, solo nos falta el fichero **paletaMSX.png** para que podamos importarlo, en el fichero **TUTORIAL5.rar** tenéis este fichero para utilizarlo. Este PNG que he creado contiene un pixel de cada color de la paleta del MSX en el orden correcto de número de color, para que el nMSXtiles sepa a la hora de importar a que color pertenece cada tinta.

Ahora descomprime el nuevo **nMSXtiles 0.6.4** que tienes comprimido en el fichero **TUTORIAL5.rar** con el nombre de **nmsxtilesv0_6_4.zip** en la carpeta C:\MSX donde tenemos el resto de herramientas. Esta es una versión Beta y tiene menos estabilidad que la versión anterior debido a otras mejoras que el programador ha implementado para arrastrar los tiles desde los bancos al screen pero que de momento he podido comprobar que hay veces que se me cuelga el programa aunque puede ser porque uso Windows 7 64 Bits, aunque lo estoy utilizando porque es muy útil para extraer tiles y bancos y demás.

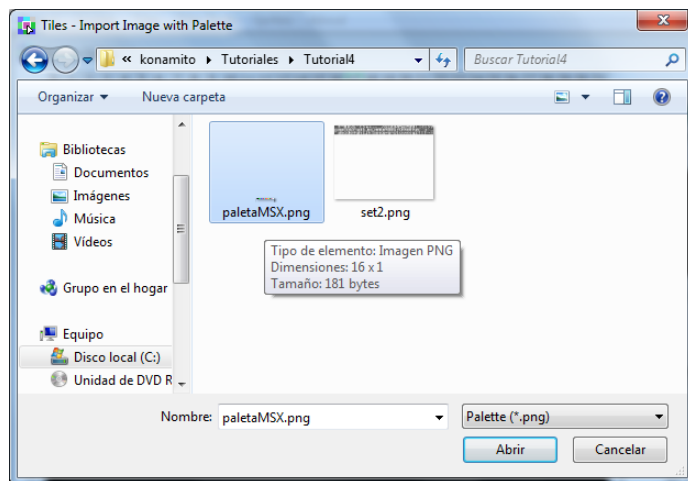


Ejecuta el **nMSXtiles** y vamos a crear un proyecto nuevo. En el menú pulsa **Project – New** veras que se muestra como en la imagen todo vacío.

Ahora vamos a importar la imagen PNG del set de caracteres creado con el GIMP. Pulsa en el menú **Tiles - Import**



Veras que lo primero que te pide es el fichero con la paleta PNG Pulsa el botón OK.

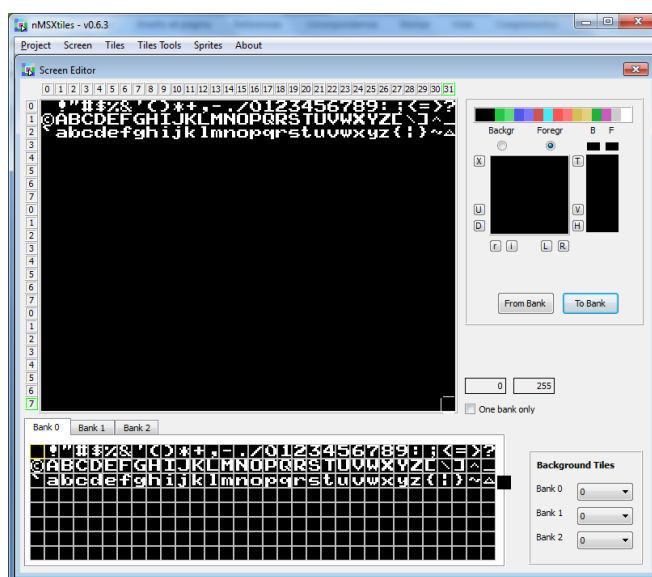


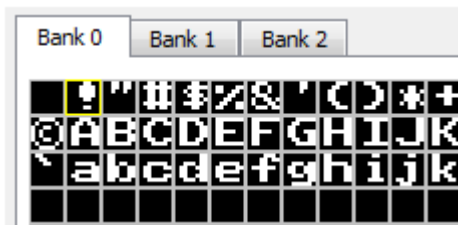
Busca el fichero **paletaMSX.png** Suministrado con los ejemplos de este tutorial. Y pulsa el botón Abrir.

A continuación nos pide el fichero de imagen selecciona el fichero **set2.png** y pulsa de nuevo el botón Abrir.

Y voila... Ya tenemos importado nuestro grafico creado con el GIMP o con el programa de gráficos que tu prefieras usar como photoshop, paint etc.

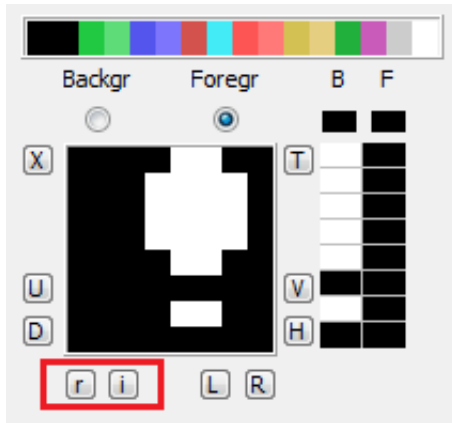
Ahora vamos a ver otros aspectos que hay que tener en cuenta a la hora de importar una imagen o grafico a nuestro proyecto en nMSXtiles.





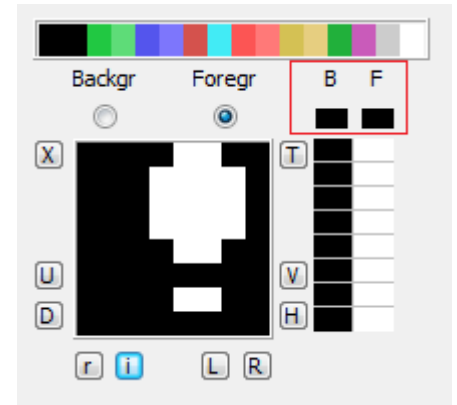
En la ventana de Bancos pulsa doble clic sobre el símbolo de admiración, con recuadro amarillo.

Todo esto te lo explico para que lo sepas para futuras importaciones de tus gráficos al nMSXtiles en blanco y negro.

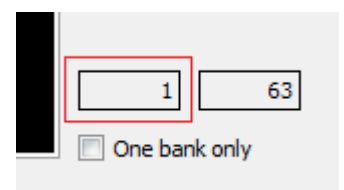


Aquí en la ventana de edición del CHR puedes ver el símbolo de la admiración que tiene tinta blanca "B" para el fondo y tinta negra para las letras "F" invertido el color ([porque ha pasado esto?](#)) el importador ha visto que hay 2 tintas pero no sabe distinguir cual es el color del fondo y cuál es el color de la tinta cambiando el orden pero dejando el mismo resultado. Puedes dejarlo tal cual esta pero deberás tenerlo en cuenta a la hora de pintar en la pantalla. Yo lo modifico, así que pulso en el botón pequeño "i" para invertir el orden de los colores. Cuando la importación mezcla colores blanco en "B" y "F" puedes pulsar en el botón "r" para reordenar todos los colores a un lado y opcionalmente pulsar después en "i" para dejarlos en "B" o "F" Después deberás pulsar en el botón [To Bank](#) para llevar el carácter modificado al banco, y repetirlo para todos.

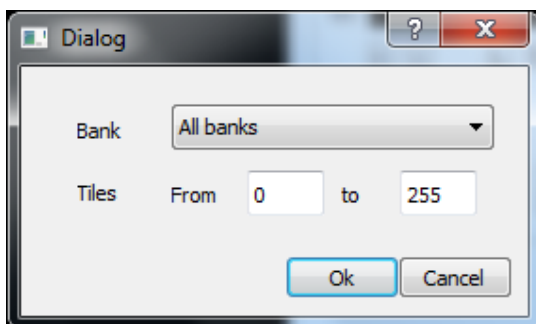
Puedes también mejorar otra cosa que la importación no tiene el cuenta si te fijas entre el cuerpo de la admiración y el punto no hay color blanco al igual que el ultimo byte del CHR, esto es normal porque al no haber pixeles no ha tenido en cuenta el color en esos bytes. Para que a la hora de comprimir no queden bytes en blanco y otros en negro y pierdas unos bytes en la compresión puedes pulsar en ese cuadrado negro justo debajo de la "F" que remarco en color rojo y pulsar a continuación en el color blanco, con esto le indicamos que todos los bytes de "B" o "F" sean de un mismo color. Pero ya que estas puesto a modificar esto después de estos pasos que te he explicado ya podrías poner un color especial a estas letras tu mismo. Te explico otro truco puedes poner una vez los colores en un CHR darle a [CONTROL+C](#) para que copie ese CHR, te sitúas en el siguiente CHR y con [CONTROL+G](#) puedes pegar solo el color ahorrándote tener que cambiar los 8 bytes en cada CHR. Así que tú mismo.



Vamos con una de la nuevas opciones agregadas a esta versión, extraer a fichero binario un número determinado de CHRs, para eso necesitamos saber el número de carácter inicial y el numero de carácter final, esto lo puedes ver en el nMSXtiles pulsando sobre el CHR y en la imagen que te pongo a la derecha de este texto. y que he remarcado en rojo. El primero es el CHR 0 y el ultimo es el CHR 95 con estos datos podemos ir a extraerlos.



Pulsa en el menú [Tiles - Export bin data...](#) En el cuadro de dialogo que se abre pon como nombre [set2](#)



Veras que sale esta ventana emergente donde te deja seleccionar que numero de banco de los 3 que tenemos y de que numero de CHR a que numero de CHR quieres extraer.

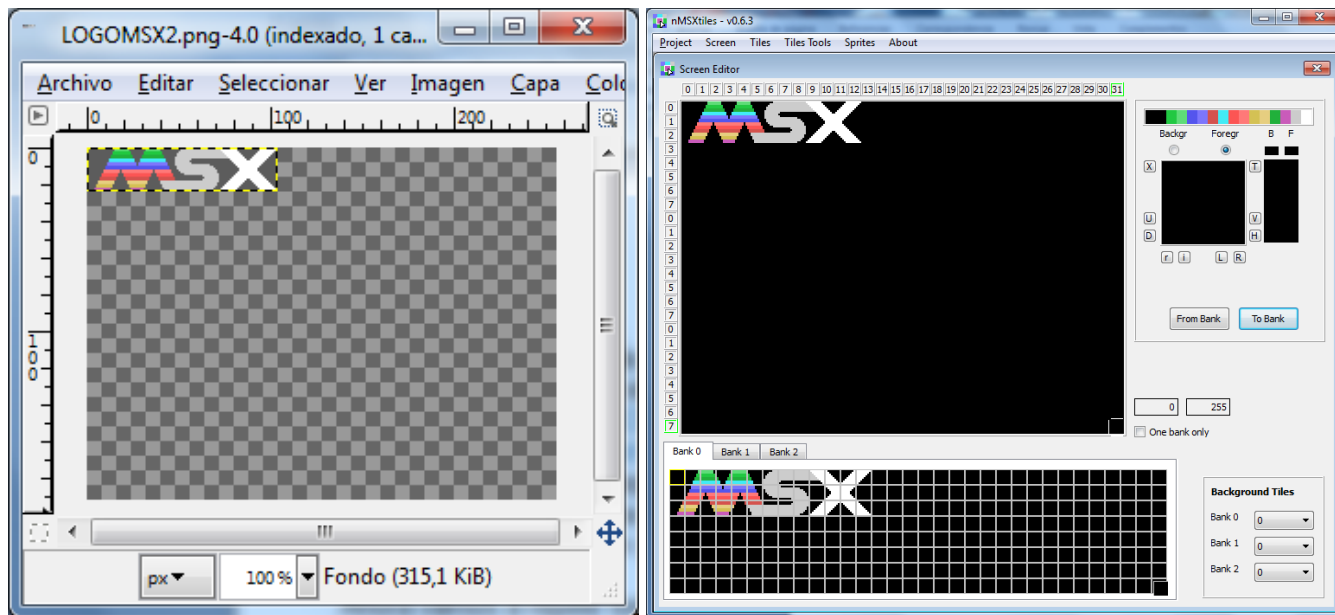
Así que selecciono el [Bank 0](#) y en Tiles selecciono From [0](#) to [95](#) y pulsamos el botón OK. Ahora nos ha guardado en el directorio 2 ficheros [set2.til](#) y [set2.col](#) que son el fichero de CHRs y los CLR de los CHRs en formato binario.

Podrías comprimirlos con Pletter como te he enseñado, Y convertirlos con binDB.exe a formato texto en ASM, Aunque no hace falta que lo hagas.

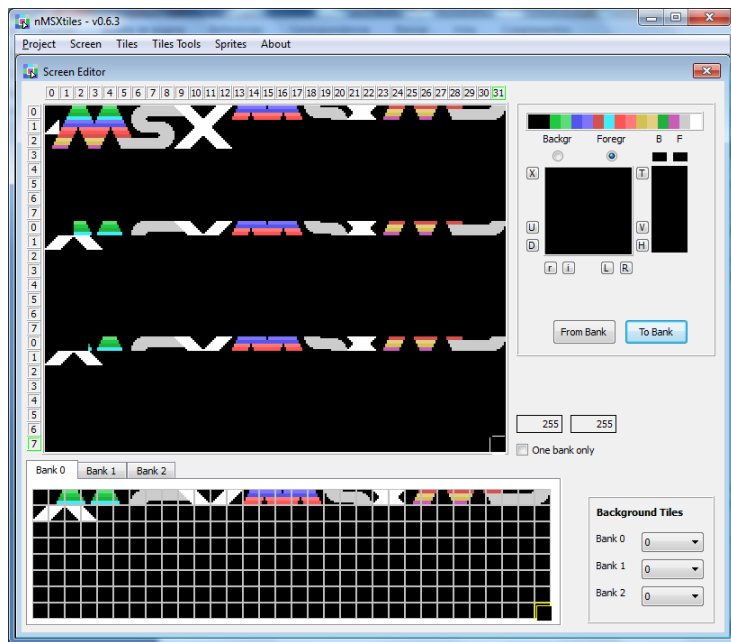
Sigamos explicando más cosas sobre el nMSXtiles

Ahora vamos a guardar el proyecto que se compone de 3 ficheros, uno el propio proyecto “.PRJ”, otro el fichero de la pantalla “.SCR” y el ultimo el fichero con los 3 bancos de CHRs con sus colores. “.TIL”
Pulsa en el menú **Project – Save** lo primero que nos pide es el nombre de la pantalla, te voy a dar un consejo, yo creo una carpeta con el nombre **Proyecto_set2** y pongo el mismo nombre para los 3 ficheros, de esta manera se que esos ficheros pertenecen a ese proyecto. Así que pongo a los 3 **set2** sin especificar extensión ya que la pone el propio programa, grabándolo en la carpeta que he creado.

Ahora vamos a repetir todo el proceso desde final de la página 2, para importar el logoMSX al nMSXtiles, ya que lo tienes abierto pulsa en **Project - New** y minimízalo, ahora abre el GIMP con el fichero **LOGOMSX2.PCX** y sigue los mismos pasos que desde el final de la pagina 2. Que son asegurarte que esta seleccionada la paleta del msx, cambiar el tamaño del lienzo, grabarlo en formato PNG unificando las capas y listo. Ahora maximiza el nMSXtiles que dejaste minimizado.



Y realiza los mismos pasos como te enseñe a partir de la pagina 3, Menú **Tiles - Import**, selecciona el fichero con la paleta PNG, y a continuación la imagen del **logoMSX2.png**, una vez que la veas cómo se muestra en las capturas que te pongo encima, asegúrate que el orden del color está bien o puedes dejarlo tal cual que no pasa nada, el resultado en pantalla será el mismo sin cambiar colores.

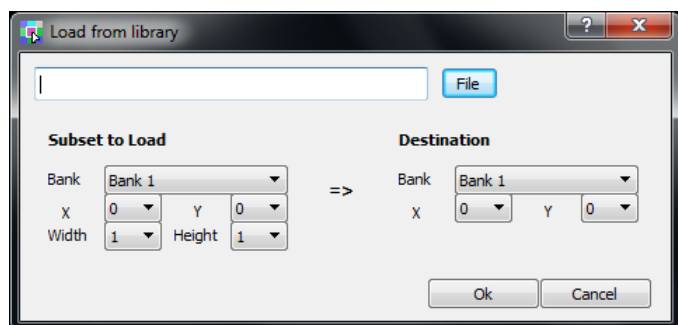


Ahora vamos a mover o poner todos los caracteres juntos o seguidos, ya que tenerlos de esta forma no es lo normal para poder exportarlos de tal número a tal número. Puedes hacerlo de forma manual o hacerlo con una opción que tiene el programa. En el menú **Tiles Tools - Group Tiles** puedes ver en la imagen que los tiles se han agrupado, ahora vamos a grabar el proyecto, crea una carpeta **Proyecto_logoMSX** y graba los 3 ficheros con el nombre de **logoMSX**.

Vamos a ver otras opciones del nMSXtiles, así que crea un nuevo proyecto.

Ahora te voy a enseñar como importar tiles sueltos de otros bancos o ficheros, para esto tienes que saber de qué fichero quieres cogerlos y el numero de tiles que te interesa importar. Con esto vamos generar el banco de CHRs que utilizaremos después para el [HolaMundoGrafico4.asm](#).

Pulsa en el menú del nMSXtiles en [Tiles – Load from library...](#) veras esta ventana emergente.



Pulsa en el botón [File](#) y selecciona el fichero [set2.til](#) que contiene los 3 bancos de tiles de la carpeta [Proyecto_set2](#)

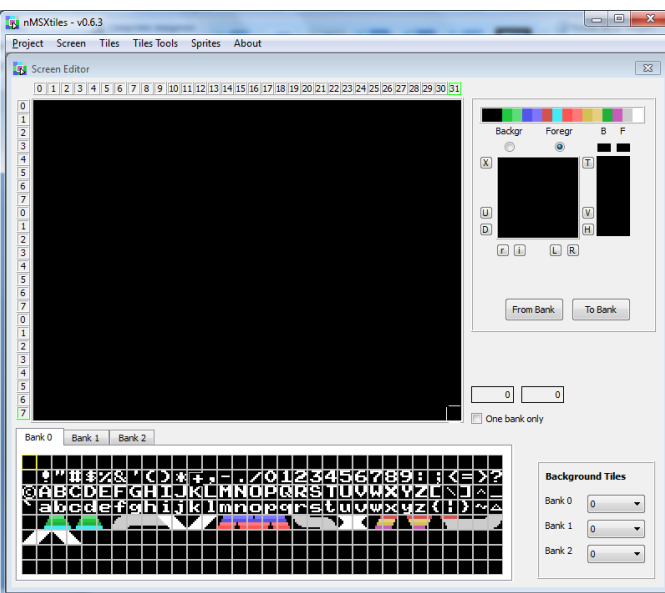
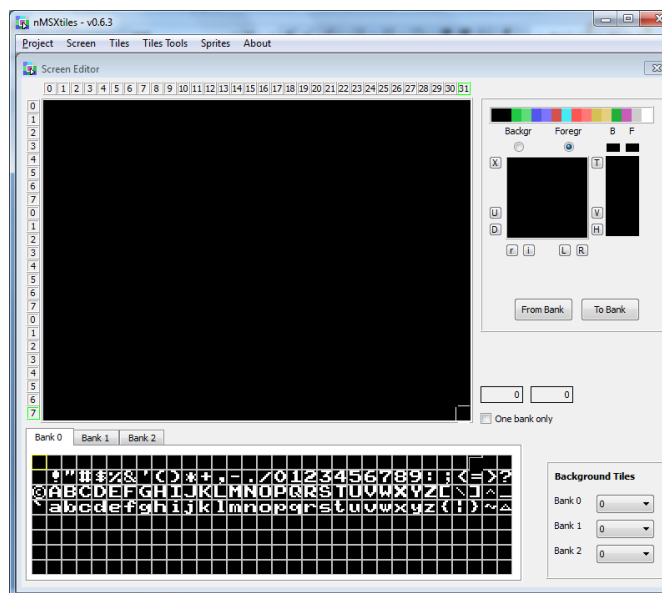
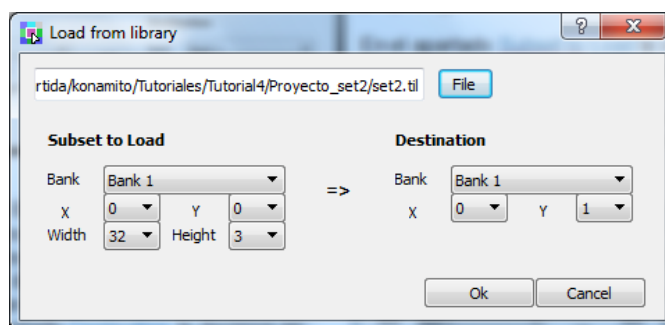
En el apartado [Subset to Load](#) le indicamos del fichero que hemos cargado tres cosas.

- 1- de cuál de los 3 bancos vamos a leer, 2- en que CHR vamos a empezar diciendo la X y la Y, y
- 3- que ancho y alto de CHRs queremos leer.

Quiero leer del Banco 1 “[Bank 1](#)”, “[X=0 e Y=0](#)” porque queremos empezar a leer desde el CHR 0, ancho le decimos “[Width 32](#)” que es el ancho del banco y alto le decimos “[Height 3](#)” que son las 3 filas que ocupan las letras en el banco.

Ahora en el apartado [Destination](#) le decimos en que banco queremos dejarlo del proyecto actual que tenemos abierto, y que hemos creado nuevo por eso esta vacio. Aquí le digo Banco 1 “[Bank 1](#)” pero como quiero situar las letras del set de

caracteres a partir del nº de CHR [32](#) le digo en “[X=0 e Y=1](#)” piensa que X e Y son la Columna y la Fila dentro del banco. Como el CHR 32 empieza en la columna 0, fila 1 debes saber también que el cuenta desde 0 a 31 la columna y de 0 a 7 la fila de cada banco. Por eso pongo X=0 e Y=1. Pulsa el botón [OK](#). Veras la imagen debajo de este texto a la izquierda si lo has hecho bien.

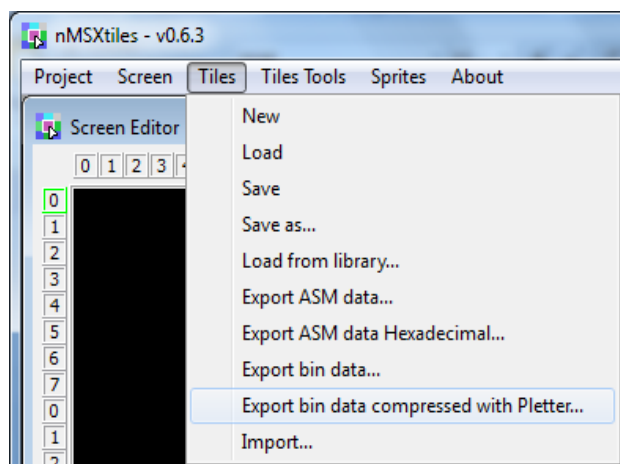


Ahora vamos a repetir el mismo proceso pero para importar los tiles de logo del MSX, supongo que ya sabrás hacerlo, te lo explico rápido menú [Tiles-Load from library](#) leo el fichero [logoMSX.til](#) de la carpeta [Proyecto_logoMSX](#) en el apartado [Subset to Load](#) pongo “[Bank 1](#)” y en “[X = 0 e Y = 0](#)” en “[Width 32](#)” y en “[Height 2](#)” en el apartado [Destination](#) pongo “[Bank 1](#)” en “[X = 0 e Y = 4](#)” y pulso el botón [OK](#).

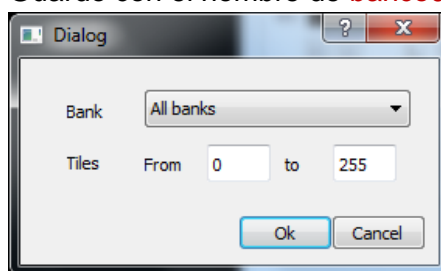
El resultado tiene que ser la imagen de la derecha encima de este texto si todo esta correcto.

Bien ya tenemos nuestro banco 0 preparado para nuestro nuevo ejemplo [HolaMundoGrafico4.asm](#) que veremos más adelante, pero antes nos queda por ver un par de cosas más del nMSXtiles.

El bank 0 está listo pero si pulsas en las pestañas del bank 1 y del bank 2 veras que están vacios, como lo que necesito es tener lo mismo en los 3 bancos, seguro que ya estas pensando en importarlo a los otros 2 bancos, pues no es así. Es más fácil de lo que piensas el propio nMSXtiles tiene un CheckBox llamado **One bank only** justo debajo de donde miras el numero de caracteres. Si marcas y desmarcas este CheckBox lo que hace es copiar el banco 0 sobre el banco 1 y el banco 2. De esta manera tendrías lo mismo en los 3 bancos puedes pulsar en los bancos para verlo, pero si estas creando menús es mas cómodo dejarlo puesto porque así no tendrás que estar cambiando de banco cuando pases de un tercio a otro de la pantalla. Lo siguiente es grabar los 3 ficheros del Proyecto con el nombre de **tutorial5** en una nueva carpeta llamada **Proyecto_tutorial5**.



Otra cosa muy importante y muy cómoda es que podemos directamente exportar los bancos de CHRs o tiles a formato binario comprimido con Pletter directamente desde el nMSXtiles, Asi que voy al menú en **Tiles - Export bin data compressed with Pletter...** Guardo con el nombre de **banco0.plet**

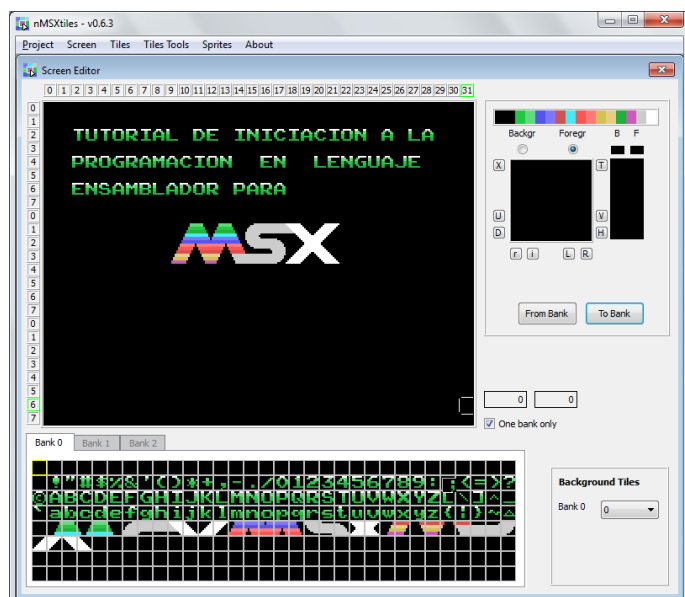


Selecciona
Bank 0

y Tiles
From **0** to **255**

Y botón OK

Ahora nos ha creado 2 fichero comprimido con Pletter con los CHRs y CLRr los fichero se llaman **banco0.plet.til** y **banco0.plet.col** y que tenemos ya listos para usar en nuestro código.



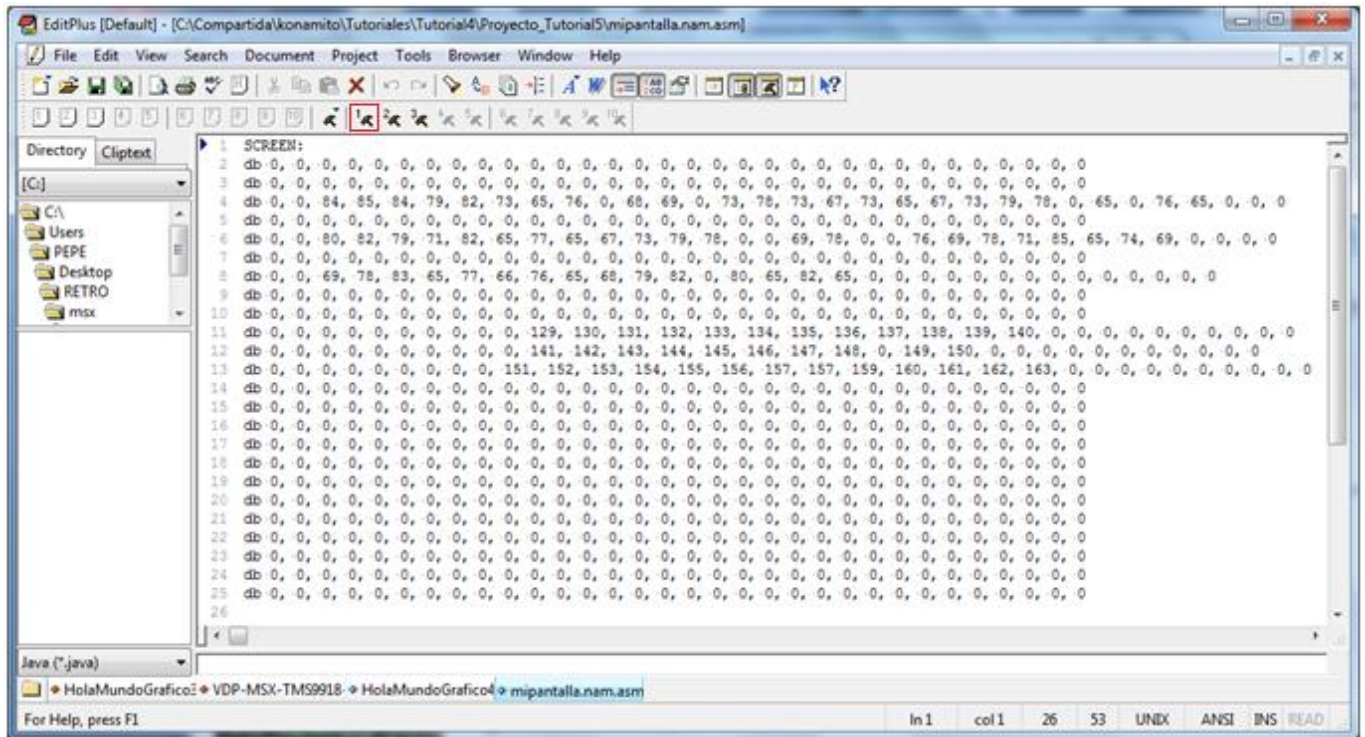
Ahora puedes ir colocando el texto y el logo del MSX o bien como lo pongo en esta imagen, o hazlo libremente a tu manera poniendo lo que quieras para que veas como es el proceso de creación de una pantalla.

Se trata de arrastrar o señalar el CHR que quieres usar del banco de CHRs y pulsar sobre la zona de la pantalla para situarlo en esa posición.

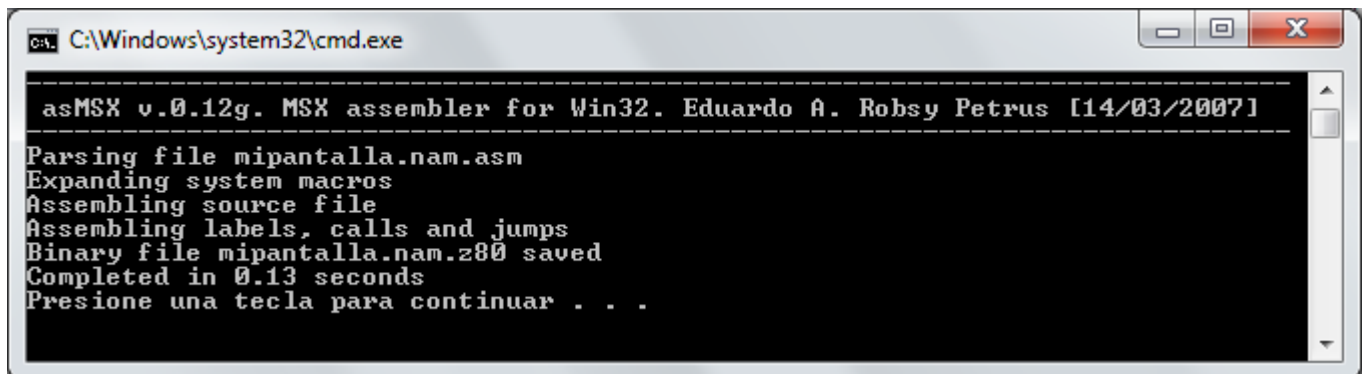
Una vez que tengas creada la pantalla vamos a grabarla comprimida en formato Pletter, para eso pulsa en menú **Screen – Export bin data compressend with Pletter...** En el cuadro de dialogo para salvar el fichero guardalo dentro del mismo proyecto del tutorial5 y dale el nombre **mipantalla.nam.plet** de esta manera sabes que es mipantalla el **“.nam”** para que sepamos que es para la NAMTBL y **“.plet”** para indicar que esta

comprimida con pletter. En mi caso la pantalla ocupa 768 Bytes, pero comprimida con Pletter se ha quedado en 129 Bytes, menos que todo el código que haría falta para crearla entre texto y código. Y sino fijate en el código nuevo cuando creamos el **HolaMundoGrafico4.asm**

Ahora ya tenemos todo lo que nos hace falta para crear el **HolaMundoGrafico4.asm**. Pero voy a explicarte otra cosa más que puede ser interesante para más adelante. Igual que puedes grabar la pantalla en binario o binario comprimido con Pletter, también puedes grabar la pantalla en formato ASM en DB's, esto puede ser útil si quieres agregarla al código, o bien cuando no quieres usar toda la pantalla completa porque tienes un marcador en un tercio o zonas de la pantalla que no usas. Para grabarla en formato DB's solo tienes que ir al menú **Screen – Export ASM data...** y en el cuadro de dialogo que se abre pones **mipantalla.nam.asm** y la grabas en el directorio del proyecto. Ahora vete a ese directorio y abre el fichero con el EditPlus, primera imagen de la siguiente pagina, donde puedes ver los 768 Bytes compuestos de 32 chrs de ancho x 24 CHRs largo de alto.



Como puedes ver es fácil eliminar líneas enteras de esta pantalla ya que es texto modificable. Esto lo utilizo en mi juego JumpinG para eliminar las 2 primeras líneas que es donde tengo mi marcador, además de añadir aquí las tablas de movimientos de los enemigos. Pero volvamos a lo que te explicaba una vez que has eliminado las líneas que no te interesan y la etiqueta **SCREEN:** cómo puedo volver a pasar a binario este fichero, pues muy fácil, pulsa el botón compilar en EditPlus remarcado en rojo.



Aquí puedes ver que el asMSX ha generado un fichero binario llamado **mipantalla.nam.z80** que podrías comprimir con Pletter y agregar a tu código, con un **.incbin** "NombreDelFichero"

Cierra el nMSXtiles y vamos con el EditPlus 3 crea un nuevo fichero y pega dentro este código.

```

;-----
; Nombre de nuestro programa
; Hola Mundo Grafico 02/02/2012
; Versión 4
;-----

;-----
; CONTANTES
;-----
; No definimos ninguna constante

;-----
; VARIABLES DEL SISTEMA
;-----
; Direcciones de la VRAM
    CHRTBL    equ    0000h    ; Tabla de caracteres
    NAMTBL    equ    1800h    ; Tabla de Nombres
    CLRTBL    equ    2000h    ; Tabla del color de los caracteres
    SPRATR    equ    1B00h    ; Tabla de los atributos de los sprites
    SPRTBL    equ    3800h    ; Tabla de Sprites

```



```

; Variables del Sistema MSX
    CLIKSW      equ    $F3DB ; Keyboard click sound
    FORCLR      equ    $F3E9 ; Foreground colour

;-----
; DIRECTIVAS PARA EL ENSAMBLADOR ( asMSX )
;-----
    .bios       ; Definir Nombres de las llamadas a la BIOS
    .page 2     ; Definir la dirección del código irá en 8000h
    .rom        ; esto es para indicar que crearemos una ROM
    .start INICIO ; Inicio del Código de nuestro Programa
; Seguir la norma del Standard MSX para una ROM
    dw 0,0,0,0,0 ; 12 Bytes a 0

;-----
; INICIO DEL PROGRAMA
;-----
INICIO:
    call    INIT_MODE_SCx      ; iniciar el modo de pantalla x
    call    INIT_GRAFICOS      ; colocar los gráficos en VRAM

; Colocar la NAMTBL comprimida en VRAM
    ld      hl,MIPANTALLA      ; Origen bytes de la NAMTBL
    ld      de,NAMTBL          ; destino NAMTBL
    call    DEPLET              ; descomprimir en VRAM

FIN:
    jp      FIN                ; esto es como 100 goto 100 para que se quede en un bucle sin fin.

;-----
;-----
; INICIALIZA EL MODO DE PANTALLA Y COLORES
;-----
; BASIC: COLOR 15,1,1
; Establecer los colores
INIT_MODE_SCx:
    ld      hl,FORCLR          ; Variable del Sistema
    ld      [hl],15            ; Color del primer plano 15=blanco
    inc     hl                 ; FORCLR+1
    ld      [hl],1             ; Color de fondo 1=negro
    inc     hl                 ; FORCLR+2
    ld      [hl],1             ; Color del borde 1=negro
; call INITXT ; BIOS set SCREEN 0
; call INIT32 ; BIOS set SCREEN 1
    call    INIGRP             ; BIOS set SCREEN 2
; call INIMLT ; BIOS set SCREEN 3
;
; SCREEN 0 : texto de 40 x 24 con 2 colores
; SCREEN 1 : texto de 32 x 24 con 16 colores
; SCREEN 2 : gráficos de 256 x 192 pixeles con 16 colores
; SCREEN 3 : gráficos de 64 x 48 pixeles con 16 colores

; Esto es para quitar el sonido que emite el msx cuando se pulsa una tecla
    xor     a                  ; ld a,0
    ld      [CLIKSW],a         ; Variable BIOS desactivar sonido teclas

; salir de la rutina INIT_MODE_SCx
    ret

;-----
;-----
; COLOCA LOS GRAFICOS EN LA VRAM
;-----
INIT_GRAFICOS:
; Esto lo realizamos para que no se vea nada en la pantalla
; Mientras colocamos los CHR y los Colores en la VRAM
    call    DISSCR             ; BIOS deshabilitar la pantalla

; Borrar los 768 Bytes de la Name Table - NAMTBL en VRAM
    ld      hl,NAMTBL          ; Dirección origen en VRAM
    ld      bc,768             ; n° de bytes a rellenar 32 columnas * 24 lineas=768 Bytes
    xor     a                  ; a=0 - Valor a rellenar
    call    FILVRM             ; BIOS -Fill block of VRAM with data byte

; aquí vamos a colocar el banco de CHRs en los 3 tercios en VRAM en la CHRTBL
; tercio 1
    ld      hl,banco0_CHR      ; Origen bytes de los CHRs
    ld      de,CHRTBL          ; destino CHRTBL tercio 1
    call    DEPLET              ; descomprimir en VRAM

```

```

; tercio 2
ld    hl,banco0_CHR      ; Origen bytes de los CHR's
ld    de,CHRTBL+2048     ; destino CHRTBL tercio 2
call  DEPLET             ; descomprimir en VRAM

; tercio 3
ld    hl,banco0_CHR      ; Origen bytes de los CHR's
ld    de,CHRTBL+4096     ; destino CHRTBL tercio 3
call  DEPLET             ; descomprimir en VRAM

; aqui vamos a colocar el banco de CLR's en los 3 tercios en VRAM en la CLRTBL
; tercio 1
ld    hl,banco0_CLR      ; Origen bytes de los CLR's
ld    de,CLRTBL          ; destino CLRTBL tercio 1
call  DEPLET             ; descomprimir en VRAM

; tercio 2
ld    hl,banco0_CLR      ; Origen bytes de los CLR's
ld    de,CLRTBL+2048     ; destino CLRTBL tercio 2
call  DEPLET             ; descomprimir en VRAM

; tercio 3
ld    hl,banco0_CLR      ; Origen bytes de los CLR's
ld    de,CLRTBL+4096     ; destino CLRTBL tercio 3
call  DEPLET             ; descomprimir en VRAM

; Esto lo realizamos para que se muestre de nuevo la pantalla.
call  ENASCR             ; BIOS habilitar la pantalla

; Salir de la rutina INIT_GRAFICOS
ret

;-----

;-----
; Rutina descompresora de RAM/ROM a VRAM pletter v1.1
; XL2S Entertainment
;-----

INCLUDE "DEPLET.asm"
;-----

;-----
; Banco de CHR's comprimido con Pletter
;-----
banco0_CHR:
.INCBIN "Proyecto_Tutorial5\banco0.plet.col"
;-----

;-----
; Banco de CLR's comprimido con Pletter
;-----
banco0_CLR:
.INCBIN "Proyecto_Tutorial5\banco0.plet.col"
;-----

;-----
; mi pantalla NAMTBL comprimida con Pletter
;-----
MIPANTALLA:
.INCBIN "Proyecto_Tutorial5\mipantalla.nam.plet"
;-----

;-----
; FINAL DE NUESTRO CODIGO EN ENSAMBLADOR.
;-----

```

A estas alturas del tutorial estoy seguro que ya debes saber que hace el [HolaMundoGrafico4.asm](#), pero le daré un pequeño repaso para explicar las cosas nuevas, aunque con lo comentarios debería bastarte.

La cabecera del programa las variables y las direcciones VRAM están más que explicadas en anteriores tutoriales, creo que lo único que no he comentado es lo de seguir la normativa después de la directiva .start INICIO para que sea más compatible con la norma de una ROM hay que dejar 12 bytes a 0 hasta el comienzo de nuestro código, si esto lo quitas funcionara sin afectar a nada, pero siempre es mejor seguir las normativas del estándar.

La rutina `INIT_MODE_SCx` como siempre establece o pone el SCREEN2, añadiéndole que quite el sonido de la pulsación de las teclas. Más adelante veremos que añadiré más cosas a esta rutina.

La rutina `INIT_GRAFICOS` veras que ha cambiado muy poco de los otros ejemplos pero es fácilmente entendible, primero desactivo la pantalla para que no se vea nada mientras colocamos los CHRs y los CLRs, borro la NAMTBL, después descomprimos los bytes del banco 0 que hemos creado con el nMSXtiles a la VRAM en cada uno de los 3 tercios, esto mismo lo repetimos con el color de los CHRs para finalmente volver a activar la pantalla al haber finalizado de colocar gráficos en la VRAM.

Ahora en vez de colocar en la NAMTBL en la columna X fila Y diferentes textos, y colocar leyendo desde una tabla los CHRs de los gráficos del logoMSX en cada una de la líneas. Pasamos a descomprimir directamente la pantalla en la NAMTBL quitándonos toda esta labor.

```
; Colocar la NAMTBL comprimida en VRAM
ld    hl,MIPANTALLA      ; Origen bytes de la NAMTBL
ld    de,NAMTBL          ; destino NAMTBL
call  DEPLET             ; descomprimir en VRAM
```

Ahora te explico el ahorro de memoria que siempre es muy importante. Vamos a contar los bytes que ocupa en formato tutorial4 y el mismo ejemplo haciéndolo con el formato empleado en este tutorial.

Formato tutorial4:

Set de letras CHRs = 504 bytes, LogoMSX CHRs =153 bytes, LogoMSX CLR = 144 bytes – TOTAL 801 Bytes

Formato nuevo:

Set de letras CHRs + LogoMSX CHRs = 623 bytes- set de letras CLR + LogoMSX CLR = 181 bytes – TOTAL 804 Bytes

Puedes ver que tenemos solo 3 bytes más, pero ya tenemos puesto el multi-color de las letras.

Formato tutorial4:

Código 86 bytes+ texto 18 bytes+ tabla color 8 bytes+ tabla nanlogo 39 Bytes+ rutina copyblock 20 Bytes = TOTAL 171 Bytes

Formato nuevo (como en el tutorial4, pantalla comprimida con el texto HOLA MUNDO Grafico y el logoMSX en 1 tercio)

Código 27 Bytes + Pantalla HolaMundo.nam.plet = 81 bytes – TOTAL 108 Bytes

Como puedes ver en el formato tutorial4 hemos empleado **972 Bytes**, mientras que en este tutorial haciendo lo mismo hemos empleado **912 Bytes**, hemos gastado **60 Bytes menos**, además de facilitarnos mucho mas el trabajo reduciendo el código.

Seguimos con el ejemplo:

```
;-----
; Rutina descompresora de RAM/ROM a VRAM pletter v1.1
; XL2S Entertainment
;-----
INCLUDE "DEPLET.asm"
;-----
```

Aquí puedes ver que he quitado todo el código de la rutina descompresora, lo he grabado en un fichero nuevo al que le he dado el nombre **DEPLET.asm**. La directiva `.INCLUDE "nombre_fichero"` lo que hace es incluir el código en ensamblador en este mismo punto donde está la directiva, es como si hiciéramos un paste del código de la rutina descompresora en este punto compilando nuestro código mas el código de la rutina **DEPLET.asm** de esta manera podemos separar rutinas o partes de código que sabemos que funcionan bien, y no nos hace falta ver en el código general ocupándonos espacio, o bien porque en un momento determinado nos puede hacer falta cambiar el sistema de descompresión por otro sistema, y con solo cambiar esta rutina ya tendríamos modificado el código de nuestro programa.

```
;-----
; Banco de CHRs comprimido con Pletter
;-----
banco0_CHR:
    .INCBIN "Proyecto_Tutorial5\banco0.plet.til"
;-----

;-----
; Banco de CLR = comprimido con Pletter
;-----
banco0_CLR:
    .INCBIN "Proyecto_Tutorial5\banco0.plet.col"
;-----
```


Aquí puedes ver que he quitado del código los bytes de los gráficos comprimidos con Pletter, haciendo uso de la directiva `.INCBIN` "nombre_fichero" lo que hace es incluir un fichero binario en esta parte del código, funciona como la directiva `.include` pero en vez de añadir código `.asm` lo que añade son DB's con datos, por eso antes de la directiva le añado una etiqueta `banco0_CHR:` o `banco0_CLR:` para después desde código saber en qué dirección empiezan esos bytes a los que quiero acceder. Además de ahorrarme tener que pasarlos por el programa `binDB` y pegarlo en el código, también puedes ver que añado el directorio donde tiene que leer los ficheros que están en [Proyecto_Tutorial5](#)

```

;-----
; mi pantalla NAMTBL comprimida con Pletter
;-----
MIPANTALLA:
    .INCBIN      " Proyecto_Tutorial5\mipantalla.nam.plet"
;-----

```

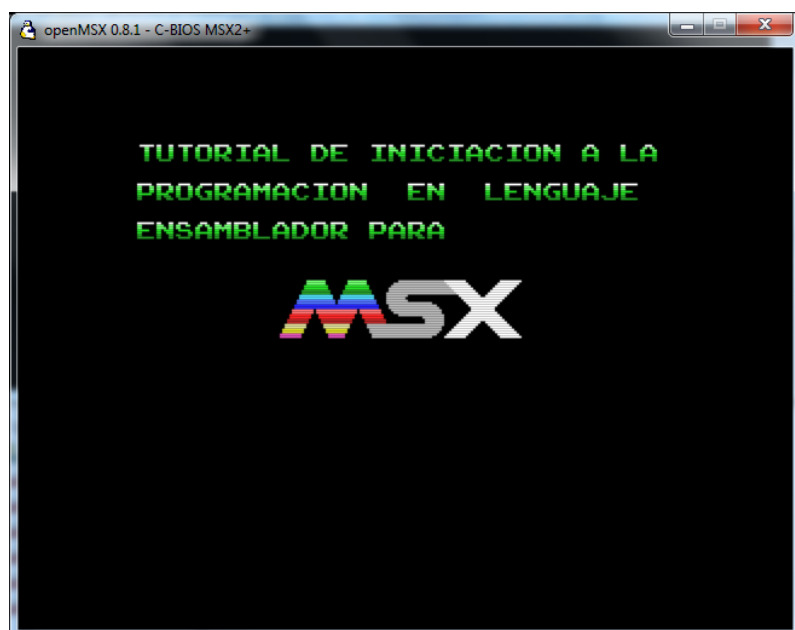
Y para finalizar también he **INCluido** un **BINario** con la pantalla que creamos con el `nMSXtiles` comprimida con Pletter al que le he añadido la etiqueta `MIPANTALLA:` para localizar la dirección. De esta manera con esta parte del código sabe que tiene que coger los Bytes de los DB's que hay en la posición de memoria que indica `MIPANTALLA` y descomprimirla en la `NAMTBL`.

```

; Colocar la NAMTBL comprimida en VRAM
ld    hl,MIPANTALLA      ; Origen bytes de la NAMTBL
ld    de,NAMTBL          ; destino NAMTBL
call  DEPLET             ; descomprimir en VRAM

```

Otra cosa que no he explicado en los demás tutoriales, es que no tenemos que indicar el número de bytes a transferir porque la rutina de descompresión Pletter tiene ya implementado en el código funciones para saber cuándo se terminan de descomprimir datos.



El resultado final será esta imagen o lo que vosotros hayas puesto.

Espero ver vuestros comentarios y vuestras capturas de pantallas, en los BLOG's o FORO's del tutorial.

Espero que haya sido de vuestro total agrado y nos vemos en el próximo tutorial. Donde explicare como se crean y como se manejan los sprites como incorporarlos al código fuente de nuestro programa. Así como los programas y herramientas que utilizo para este cometido también en varias partes.

José Vila Cuadrillero

"ES DETESTABLE ESA AVARICIA ESPIRITUAL QUE TIENEN, LOS QUE SABIENDO ALGO, NO PROCURAN LA TRANSMISION DE ESOS CONOCIMIENTOS."

Miguel de Unamuno

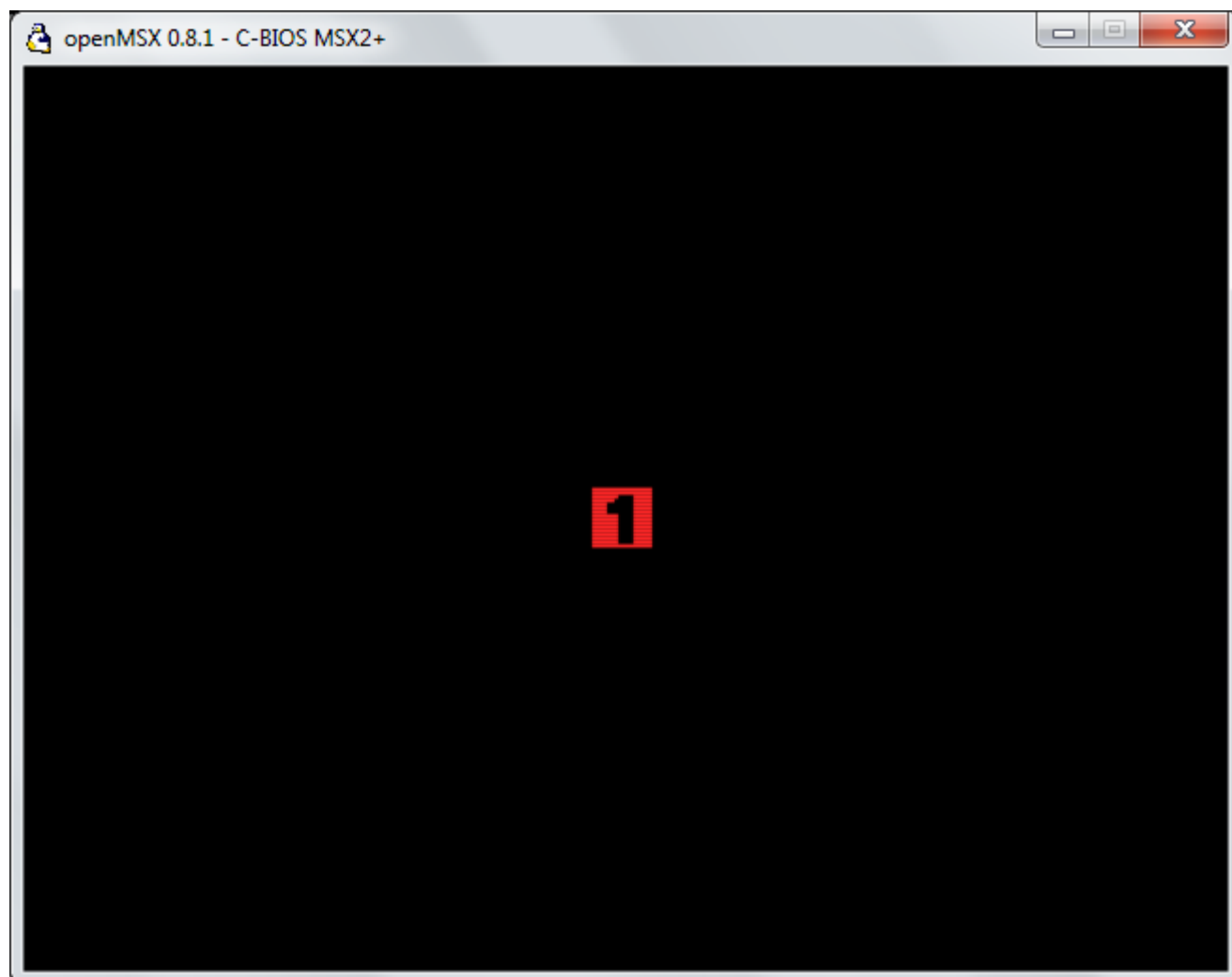
Escritor y Filósofo.

(Bilbao 1864 - Salamanca 1936)

TUTORIAL DE INICIACION A LA PROGRAMACION EN LENGUAJE ENSAMBLADOR PARA MSX

6ª PARTE – EL MUNDO DE LOS SPRITES 1

En esta parte del tutorial veremos cómo funciona el mundo de los sprites en el MSX1 crearemos varios ejemplos de código, empezando con un [Hola Sprite](#) que iremos modificando, veremos la parte de la teoría, así como las herramientas que necesitaremos para trabajar con sprites.



Antes de empezar con el código del [HolaSPR1.asm](#), vamos con la parte que menos os gusta, la teoría... ya sabéis que siempre es necesario para comprender los que queremos realizar.

Recordáis que en la tercera entrega en la pagina 4 cuando os explicaba las posiciones de la memoria VRAM deje unas direcciones sin explicar. ([Os reproduzco el fragmento.](#))

Queda la **Sprite Pattern Table - SPRTBL** y la **Sprite Attribute Table - SPRATR** que veremos más adelante en otra entrega del tutorial dedicada al mundo de los Sprites.

Pues ha llegado el momento de explicarlo. La **Sprite Pattern Table - SPRTBL** abreviado, es la zona de la VRAM comprendida entre las posiciones de memoria **3800h** a **3FFFh** ocupa **2Kb** y es donde colocamos los CHRs de nuestros SPRITES recordar que en el MSX todo funciona con CHRs de 8x8 pixeles, como tenemos 2048Kb para sprites cuantos CHRs son?, fácil $(2048 / 8) = 256$ CHRs como máximo numerados del 0 al 255, podremos almacenar en esta parte de la memoria 256 Sprites de 8x8 o 64 sprites de 16x16 pixeles, pero el hardware del MSX solo nos permite usar 32 Sprites* simultáneos en pantalla al mismo tiempo. [Y no puedo poner más de 64 sprites en VRAM?](#) siempre puedes volcar a esta zona de la VRAM más SPRs a medida que los vayas necesitando o cuando se cambia de fase.

El área que maneja los sprites en la pantalla se llama **Sprite Attribute Table - SPRATR** abreviado, es la zona de la VRAM comprendida entre las posiciones de memoria **1B00h** a **1B7Fh** ocupa **128 Bytes** aquí colocamos los Atributos de los Sprites que hacen que estos se visualicen en la pantalla. Permite abreviar estos términos de la siguiente manera ATRs son los Atributos de los Sprites y SPRs es como abrevio a los Sprites. Esta zona de la VRAM de 128 Bytes está compuesta por una **tabla de 4 valores**. Si dividimos **128** entre **4** ($128 / 4 = 32$) Que son el número máximo de sprites que podemos usar al mismo tiempo en la pantalla. Como se compone esta tabla te la explico a continuación

1º Valor – Coordenada Y - Esta es la coordenada Y del SPR su valor va desde 0 hasta 255.

2º Valor – Coordenada X - Esta es la coordenada X del SPR su valor va desde 0 hasta 255

3º Valor – Numero de CHR – que CHR dentro de la SPRTBL usaremos para el SPR de 0 a 255

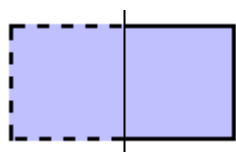
4º Valor – Numero de Color – Aquí le decimos 2 cosas: 1-Que color de 0 a 15 de la paleta del MSX.

La segunda cosa en el Cuarto valor se conoce como EC o Early Clock si activamos el bit 7 de este byte lo que hace es restar 32 pixeles a la coordenada X del SPR, (**Sigo sin entender esto**)

Si os fijáis en muchos juegos cuando el personaje sale por el lado derecho de la pantalla desaparece poco a poco o pixel a pixel (**No en todos**) Sin embargo cuando lo hacen por el lado Izquierdo no sucede esto en la mayoría de juegos.



Esto sucede porque cuando el SPR llega a la coordenada X 0 esta pasa automáticamente a la 255 apareciendo el SPR por el lado derecho de la pantalla, para que esto no suceda, si activamos el EC, le resta 32 pixeles a la coordenada X del SPR.



De esta manera puedes seguir restando la coordenada X para que se produzca una salida por el lado izquierdo igual que lo hace por el lado derecho, pero tenemos que estar gestionando por software el control de esta opción que suele ser muy engorrosa por eso en casi todos los juegos omiten hacerlo. Yo incluido.

Hay cuatro formas de definir los SPRs: 2 normales y 2 agrandados o ampliados.

1 – Sprites de 8x8 normales

2 – Sprites de 8x8 agrandados

3 – Sprites de 16x16 normales

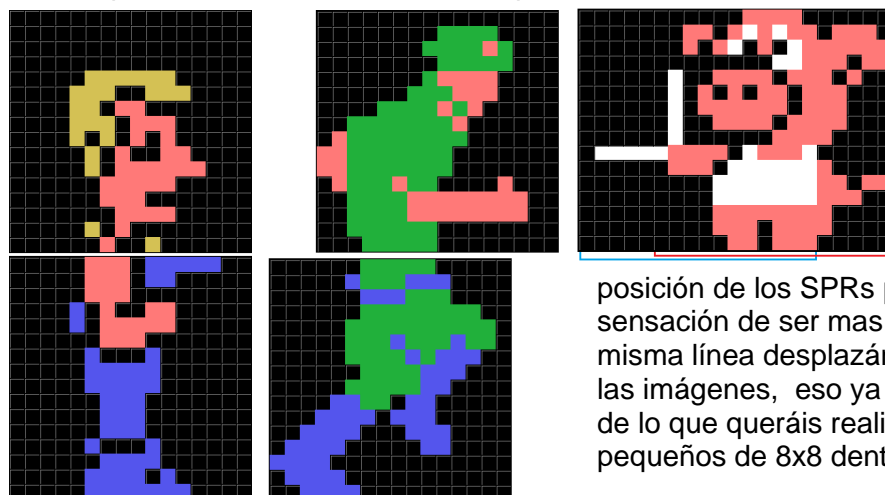
4 – Sprites de 16x16 agrandados

Si activamos la opción del VDP para agrandar los sprites lo que hace es doblar el número de pixeles el tamaño no cambia sino que se amplía un SPR de 16x16 pasa a ocupar en la pantalla 32x32 y uno de 8x8 pasa a ocupar 16x16, pero no cambiamos los CHRs de los sprites sino que se hace un zoom.

Solo podemos definir al mismo tiempo en la pantalla un formato de SPRs de los 4 disponibles.

Entonces si uso SPRs de 16x16 normales, no puedo usar sprites de 8x8 o más grandes de 16x16?

Esto no es exactamente así. Si se selecciona este formato de SPRs puedes crear diferentes tamaños dentro de un SPR de 16x16 aunque sea de 8x8 pero después desde código tendrás que tener en cuenta el tamaño que has creado para el SPR, incluso se pueden usar 2 SPRs de 16x16 uno encima del otro para crear un SPR de 32x16 o puedes crearlos más anchos o mas grandes juntado varios SPR.



EJEMPLOS DE SPRs:

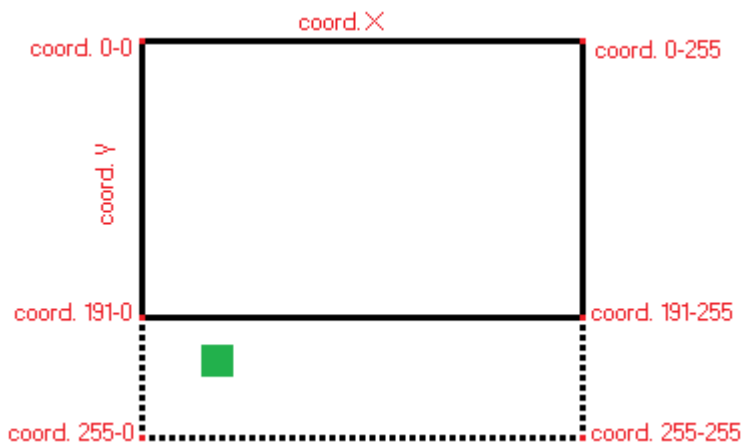
Como podéis ver no solo tenemos que limitarnos al tamaño de 16x16 se pueden hacer SPRs más grandes usando varios SPRs, y si desplazamos la

posición de los SPRs podemos conseguir que den la sensación de ser mas grandes, o los puedes poner en la misma línea desplazándolos X pixeles como puede ver en las imágenes, eso ya depende de nuestra imaginación y de lo que queráis realizar. También podrías crearlos más pequeños de 8x8 dentro de un SPR de 16x16.

Vamos a explicar con más detalles cada uno de los valores de la tabla de ATRs de SPRs.

1º Valor – Coordenada Y - Esta es la coordenada Y del SPR su valor va desde 0 hasta 255.

2º Valor – Coordenada X - Esta es la coordenada X del SPR su valor va desde 0 hasta 255



Aquí te pongo una imagen para explicártelo. La resolución del MSX en modo SC2 es de 256x192 pero el ordenador siempre cuenta desde 0, entonces es 255x191

Pero para los SPRs tiene una resolución de 255x255, como puedes ver en la imagen tenemos un SPR en la posición mas allá de la 191 que no se visualizaría en la pantalla, pero si podría estar en esa zona de la pantalla si su coordenada Y pasa de 192, en cuando superara la 255 empezaría a aparecer de nuevo al principio de la pantalla

Otra cosa que tienes que saber es que si sitúas un SPR en la [Coordenada Y 208](#) dejaran de verse todos los SPRs que tengamos por encima de este plano. Y si lo sitúas en la [Coordenada Y 209](#) dejara solo de verse ese SPR aunque no tiene mucho sentido ya que esta fuera de la pantalla y no lo vemos. Más cosas curiosas si sitúas un ATR en la [coord. Y 8](#) realmente tienes que ponerlo en la [coord. Y 7](#), al igual que si lo sitúas en la [coord. Y 0](#) este se pondrá en la [coord. Y 1](#), cosas extrañas del VDP.



Planos de los sprites

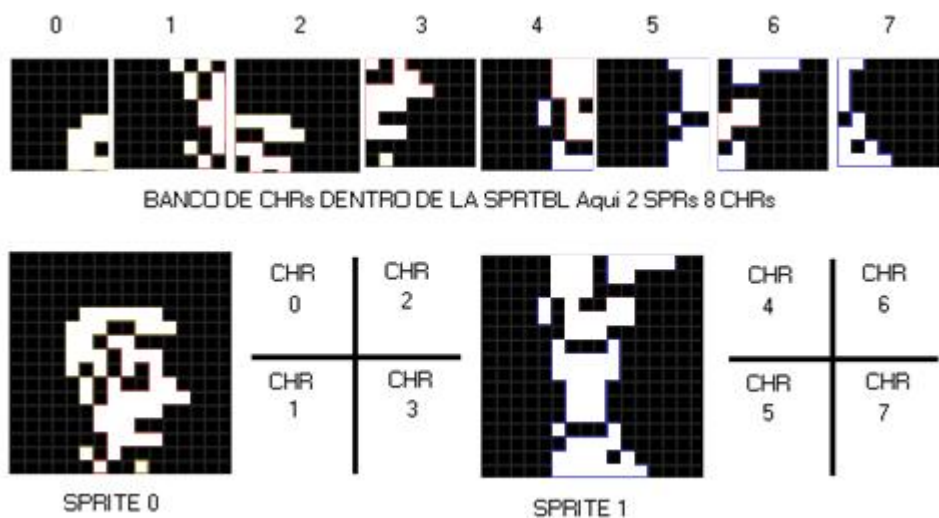
Te explico el tema de los planos para que lo entiendas cada ATR que sitúas en la SPRATR va tomando un numero consecutivo de Plano empezando por el 0 y terminando por el 31. Ahora vamos a pensar en profundidad de planos el ATR1 sería el más cercano a la pantalla y el ATR32 sería el más alejado hacia el fondo de esta, te pongo una imagen que seguro

que lo entiendes mejor, esto quiere decir que el ATR1 siempre seria el que pasa por encima de los demás sprites y así consecutivamente hasta llegar al ATR32 que sería el más alejado.

Sigamos con el tercer valor de la tabla de ATRs de SPRs.

3º Valor – Numero de CHR – que CHR dentro de la SPRTBL usaremos para el SPR de 0 a 255

Al igual que ocurre con el banco de CHRs en la CHRTBL en la [SPRTBL](#) funciona de igual manera tenemos un banco de 256 CHRs numerados del 0 al 255 que son los gráficos de nuestros SPRs. Si hemos definido el VDP para usar SPRs de 8 x 8 cada CHR será un SPR pero si hemos definido que son de 16x16 cada SPR está compuesto de 4 CHR, esto quiere decir que aquí en este valor pondríamos 0 si queremos usar el SPR0 pero si queremos usar el SPR1 aquí tendríamos de decirle 4 que es donde empiezan los CHRs del SPR1. Vamos como siempre con las imágenes que lo aclaran más.



Sigamos con el cuarto valor de la tabla de ATRs de SPRs

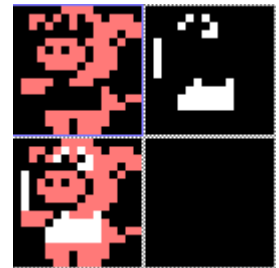
4º Valor – Numero de Color – Que color de 0 a 15 de la paleta del MSX usara el SPR.

A cada SPR solo le podemos dar un color, en verdad son 2 colores el 0 que es el color transparente este no lo puedes modificar y el propio color que le das SPR de 0 a 15 de nuestra paleta MSX.

Entonces no puedo usar más de un color para un SPR? La respuesta es NO

No podemos usar más de un color por SPR pero si podemos poner un SPR encima de otro SPR para conseguir que nuestro SPR tenga 2 o más colores. Como podéis ver en la imagen 2 SPRs con 1 color y la mezcla de los 2 SPRs.

Aunque debes saber que esto gastara 2 SPRs en la misma línea hay trucos para usar un tercer color poniendo un SPR más arriba de este SPR para que en la misma línea no coincidan y se produzca la terrible regla del 5º Sprite. (Y os preguntareis que es esto tan terrible)



Hay una limitación inherente al hardware que no podemos eliminar aunque si paliar un poco.

1 2 3 4 5



Regla del 5º Sprite

Me refiero al hecho de situar 5 SPRs en la misma línea horizontal, os recuerdo que siempre hablo de MSX1, en MSX2 esto mismo sucede pero con 8 SPRs en la misma línea. El problema es que si situamos 5 SPRs en la misma línea, el 5º SPR

desaparecerá, bueno más bien que no se visualizara en la pantalla aunque si este en ese sitio. Tengo preparado un ejercicio para demostraros esto viéndolo en la pantalla de nuestro MSX más abajo.

Y que se puede hacer para paliar esto... Pues hay 2 posibles soluciones.

1 – No utilizar más de cuatro SPRs en la misma línea horizontal.

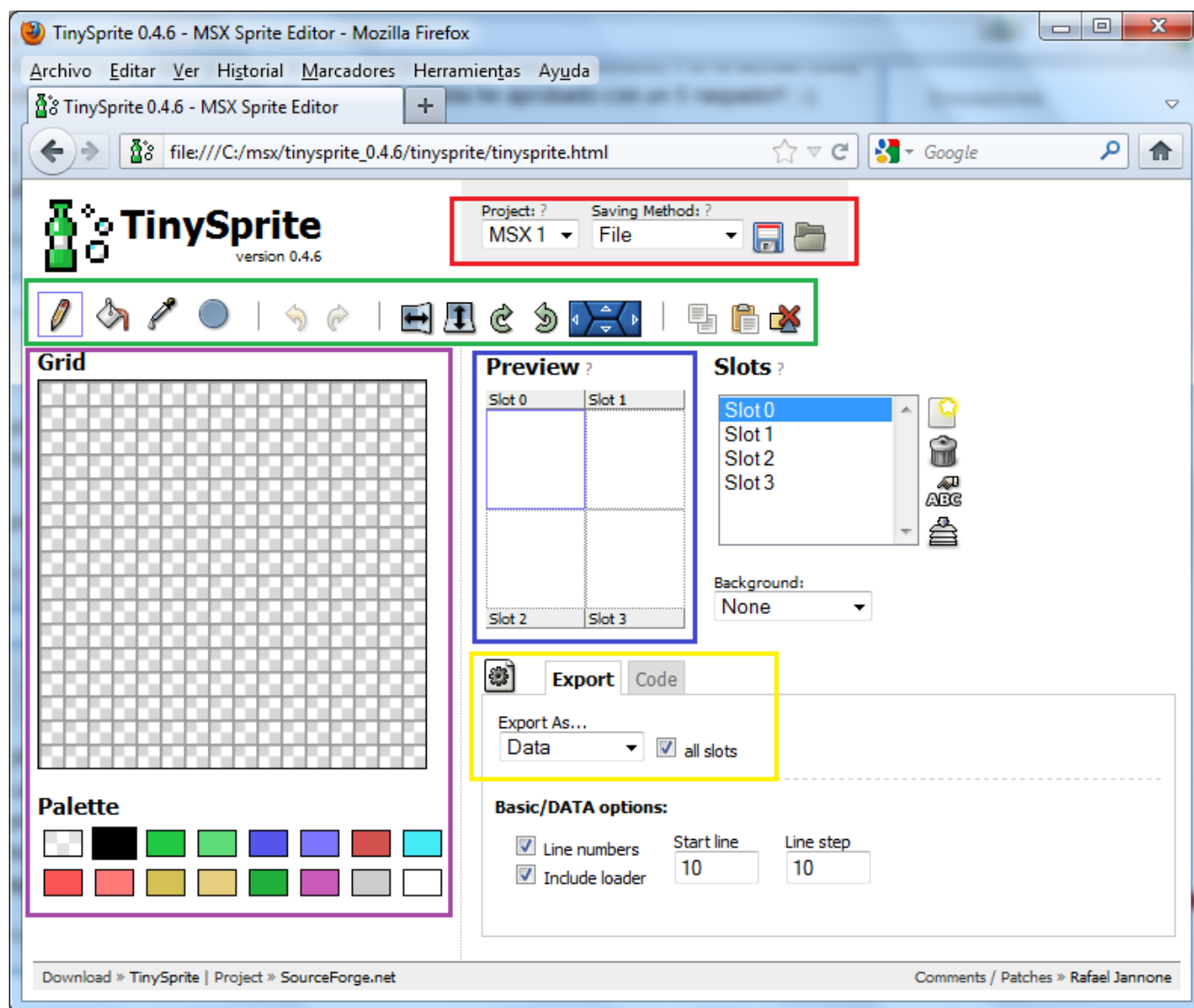
2 – Usar trucos con los SPRs y crear una rutina de Parpadeo para paliar un poco este efecto.

Si se colocan estratégicamente los enemigos a diferentes alturas teniendo en cuenta 16 pixeles de distancia, podremos fácilmente saltar esta limitación porque nunca sucederá. Hay ocasiones en los que durante unos instantes se puede dar el caso de coincidir 5 SPRs en línea, entonces tendremos que crear por código una rutina de Parpadeo-Flickering. Lo que hace esta rutina es variar el orden de los SPRs en la SPRATR modificando los planos de los SPRs el 4º será el 1º el 1º será el 2º el 2º será el 3º y el 3º será el 4º repitiendo esta rotación constantemente lo que haremos es que durante unas decimas de segundo parpadee cada vez un SPR distinto, de esta manera el 5º SPR no desaparece pero vemos un pequeño parpadeo en los 5 SPRs, hasta que dejen de estar en la misma línea. Esto puede ser útil para un instante pero no para poner 5 SPRs en línea constantemente porque el efecto es molesto.



Aquí te pongo 2 capturas de ejemplos, En Zombie Incident la prota usa 3 SPRs, uno encima de otro y un 3º en medio de los 2 pero en línea solo tiene 4 los 2 de la prota mas los 2 del zombie verde. Yo en JumpinG uso 3 SPRs para el prota mas 2 SPR enemigos de 2 perros en la misma línea, eso son 5 pero yo opto porque no se visualice el 5º SPR que es la sombra negra del prota en vez de realizar por código parpadeo-Flickeo. Hay que jugar con estas cosas para salvar este Hándicap. Eso será elección tuya.

Vamos con la parte practica que la teoría ya la hemos pasado. Antes de hacer nuestro [HolaSPR1.asm](#) tenemos que crear el SPR que usaremos para nuestro ejercicio, la herramienta que yo utilizo para crear los SPRs está en el pack-MSX de la 1ª entrega y se llama el fichero comprimido [tinysprite_0.4.6.zip](#) descomprímelo en la carpeta C:\MSX donde tienes todas las herramientas, veras que te ha creado una carpeta llamada [tinysprite_0.4.6](#) y dentro de esta carpeta otra carpeta llamada [tinysprite](#), este programa creado por Rafael Jannone esta creado en JavaScript por eso veras que el ejecutable o donde tienes que pulsar doble clic es el fichero [tinysprite.html](#) pulsa y se abrirá con tu navegador web.(No I-explorer.)



Te he puesto unos rectángulos de colores para enseñarte un poco como funciona este programa que aunque es muy simple usarlo no está de más explicarlo.

1º - Rojo - Aquí seleccionamos SPRs para MSX1 o 2 además de grabar o cargar un SPR.

2º - Verde – como en muchos programas de dibujo lápiz para pintar al pixel, relleno, leer un color, crear círculos, Undo y restore, Flip Horizontal o vertical, rotación, desplazar la ventana, copiar pegar y borrar.

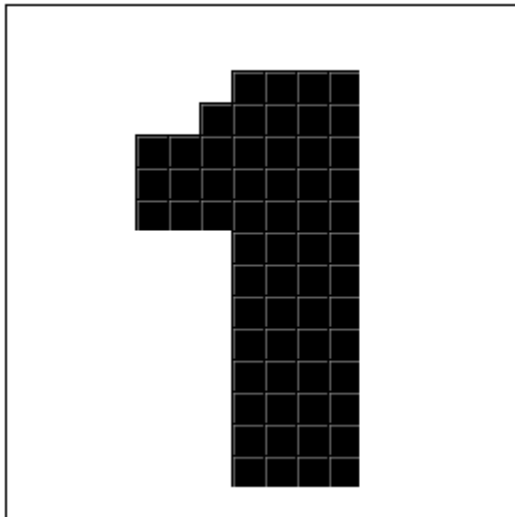
3º - Morado - Es la zona de la pantalla donde dibujamos el SPR y donde seleccionamos los colores a usar. Tengo que decirte que cada color que uses aquí seria un SPR distinto.

4º - Color Azul – Es la zona donde vemos en miniatura cada uno de los 4 posibles SPRs a crear.

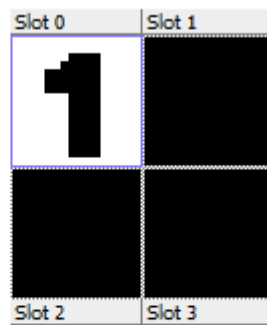
5º - Color Amarillo – Sin lugar a dudas es la zona más importante, junto con la de leer y grabar porque aquí es donde nos genera los DB's de los bytes de los SPRs a formato ASM u otros.

Vamos a crear nuestro primer SPR lo primero que selecciono es en el ComboBox “Background” el color “Black” me gusta que el fondo sea negro porque en la mayoría de los ejemplos uso este color de fondo.

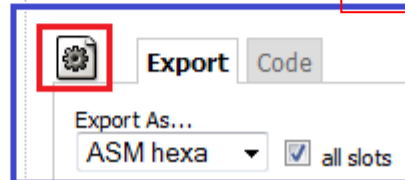
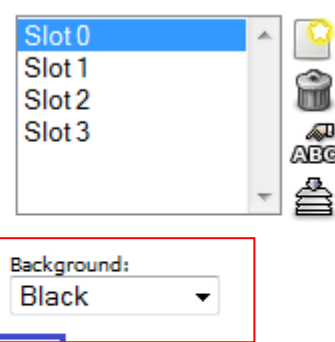
Grid



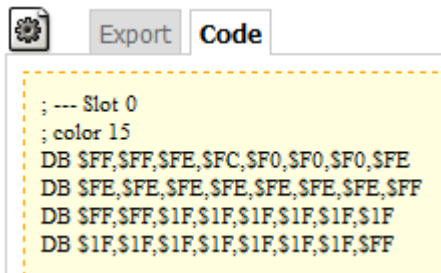
Preview ?



Slots ?

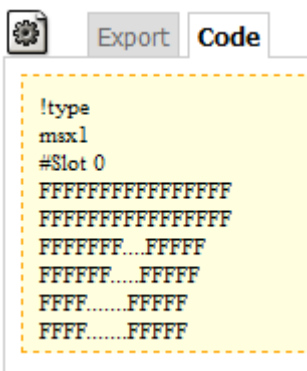


Después creo un SPR cuadrado haciendo un fill de todo el SPR en blanco y vacío con tinta transparente **NO** con negro el SPR número 1, esto es para que se vea bien el SPR de 16x16 cuadrado y en interior del número deje ver lo que hay de detrás de la pantalla. Remarcado en Azul - En el ComboBox **Export_As...** aquí seleccionamos en que formato vamos a exportar los datos. Selecciona **ASM hexa** finalmente desmarca el CheckBox que en la imagen ves con una V opción **all slots** para que solo exporte el Slot 0 y no los 4 Slots, o posibles SPRs y pulsáis sobre el recuadro en rojo botón **Export**.

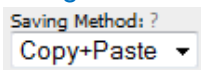


Esto es lo que nos genera el tiny que no es otra cosa que los bytes de los 4 CHR que componen nuestro primer SPR, como es texto podemos copiarlo y pegarlo en nuestro código. Datos que te da el tiny, de que Slot es el SPR, en este caso solo el Slot 0, y que color usa el SPR en este caso el 15 que es blanco a modo de comentario. Si creas un SPR con 2 CLR te duplicaría esta información como te he explicado en la teoría porque serían 2 SPRs y tu sabrías cual es un SPR y cuál es el otro porque dentro del Slot 0 te pondría 32 bytes para el 1º color y 32 bytes para el 2º color. Vamos a ver como guardar este SPR en el tiny.

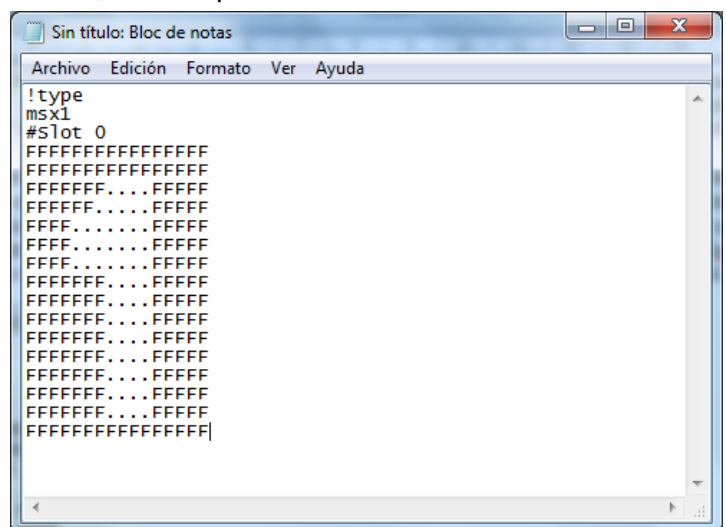
En el apartado 1º el Rojo del tinysprite tienes el botón **Save** os pongo la imagen a la derecha. Pulsando el botón y después diciéndole el nombre guardaría el fichero pero hay un problema. He de deciros que a mí no me funciona después la opción de Load y nunca puedo cargar el fichero por eso uso otra opción para grabar mis proyectos de SPRs, os lo explico.



En el ComboBox **Saving Method:**



cambio la opción de **File** por la opción de **Copy+Paste** y pulso el botón **Save** veras la imagen de la IZQ..



Ahora abro el **Bloc de notas de Windows** y copio y pego el texto del tiny al bloc de notas y guardo el fichero como **SPR1_paste.txt**

Para cargarlo de nuevo abro el tiny siempre con la opción **Copy-Paste** pulso el botón **Load** y en la ventana que se abre pego el texto del bloc de notas y pulso el botón **Import** y ya me sale mi SPR.

Vamos con el código del **HolaSPR1.asm** abrir el **EditPlus** y creáis un fichero nuevo ASM, copia y pega todo el texto a partir de esta línea en el fichero nuevo que habéis creado en el EditPlus.

```

;-----
; Nombre de nuestro programa
; Hola Sprite 1
; 17/10/2011
;-----

;-----
; CONSTANTES
;-----
        NUM_SPR      equ    1      ; Numero de SPRITES
        NUM_ATR      equ    1      ; Numero de Atributos de Sprite

;-----
; VARIABLES
;-----
; Direcciones de la VRAM
        CHRTBL       equ    0000h   ; Tabla de caracteres
        NAMTBL       equ    1800h   ; Tabla de Patrones
        CLRTBL       equ    2000h   ; Tabla del color de los caracteres
        SPRTBL       equ    3800h   ; Tabla de Sprites
        SPRATR       equ    1B00h   ; Tabla de los atributos de los sprites
; Variables de sistema
        CLIKSW       equ    $F3DB   ; Keyboard click sound
        FORCLR       equ    $F3E9   ; Foreground colour
        RGLSAV       equ    $F3E0   ; Content of VDP(1) register (R#1)
        STATFL       equ    $F3E7   ; Content of VDP(8) status register (S#0)

;-----
; DIRECTIVAS PARA EL ENSAMBLADOR ( asMSX )
;-----
        .bios        ; Definir Nombres de las llamadas a la BIOS
        .page 2      ; Definir la dirección del código irá en 8000h
        .rom         ; esto es para indicar que crearemos una ROM
        .start INICIO ; Inicio del Código de nuestro Programa
; Seguir la norma del Standard MSX
        dw 0,0,0,0,0 ; 12 ceros
; Identificacion
        db "HolaSPR1",1Ah

;-----
; INICIO DEL PROGRAMA
;-----
INICIO:
        call INIT_MODE_SCx ; iniciar el modo de pantalla
        call INIT_GRAFICOS ; imprimir el mensaje en pantalla

; Mostrar nuestro primer sprite
; Mover los ATRs de los SPRs en ROM/RAM a la SPRTBL en VRAM
        ld     hl, tblATRsSPRs ; Tabla de los ATRs de los SPRs en ROM
        ld     de, SPRATR      ; Destino la Sprite Attribute Table
        ld     bc, (NUM_ATR*4) ; Numero de Bytes
        call   LDIRVM          ; 05Ch - BIOS - Copy block to VRAM, from memory

FIN:
        jp     FIN            ; esto es como 100 goto 100

;-----
INIT_MODE_SCx:
; INICIALIZA EL MODO DE PANTALLA Y COLORES
;-----
; BASIC: COLOR 15,1,1
; Establecer los colores
        ld     hl, FORCLR      ; Variable del Sistema
        ld     [hl], 15        ; Color del primer plano 15=blanco
        inc    hl              ; FORCLR+1
        ld     [hl], 1         ; Color de fondo 1=negro
        inc    hl              ; FORCLR+2
        ld     [hl], 1         ; Color del borde 1=negro
; call INITXT      ; 06Ch - BIOS - Initialize VDP to 40x24 Text Mode - set SCREEN 0
; call INIT32      ; 06Fh - BIOS - Initialize VDP to 32x24 Text Mode - set SCREEN 1
        call   INIGRP          ; 072h - BIOS - Initialize VDP to Graphics Mode - set SCREEN 2
; call INIMLT      ; 075h - BIOS - Initialize VDP to Multicolour Mode - set SCREEN 3
;
; SCREEN 0 : texto de 40 x 24 con 2 colores
; SCREEN 1 : texto de 32 x 24 con 16 colores
; SCREEN 2 : graficos de 256 x 192 pixeles con 16 colores
; SCREEN 3 : graficos de 64 x 48 pixeles con 16 colores
;

```

```

; Esto es para quitar el sonido que emite el msx cuando se pulsa una tecla
xor     a          ; ld a,0
ld      [CLIKSW],a  ; Variable BIOS desactivar sonido teclas

; Desactivar las teclas de funcion
call    ERAFNK      ; 0CCh - BIOS - Erase function key display

; Modificar el Registro 1 del VDP ( Video Display Procesor )
;
; bit 7 : 10000000b: 1= siempre a 1 (4/16k vram)
; bit 6 : 01000000b: 1= Pantalla activada | 0= Pantalla desactivada
; bit 5 : 00100000b: 1= interrupciones activadas | 0= interrupciones desactivadas
; bit 4 : 00010000b: 1= modo texto | 0= otros modos
; bit 3 : 00001000b: 1= modo multicolor | 0= otros modos
; bit 2 : 00000000b: 0= siempre a 0
; bit 1 : 00000010b: 1= sprites de 16x16 | 0= sprites de 8x8
; bit 0 : 00000001b: 1= sprites *2 Ampliados | 0= sprites *1 no ampliados

;      ld      bc,(11100010b<<8)|1  ; BC=E201h en binario

      ld      bc,$E201      ; B=E2 en binario es 11100010b y C=01 que es el registro del VDP
call    WRTVDP             ; 047h - BIOS - Write to any VDP register
ret                                           ; salir de la rutina
;-----

;-----
INIT_GRAFICOS:
; COLOCA LOS GRAFICOS EN LA VRAM
;-----
; Esto lo realizamos para que no se vea nada en la pantalla
call    DISSCR            ; 041h - BIOS - Disable screen

; Borrar los 768 Bytes de la tabla de nombres en VRAM
ld      hl,NAMTBL         ; Origen la Name Table
ld      bc,768            ; Numero de bytes 32x24
xor     a                 ; ld a,0 - Valor a rellenar
call    FILVRM            ; 056h - BIOS - Fill block of VRAM with data byte

; Colocar los bytes de los SPRs en la Sprite Pattern Table
ld      hl,SPR1           ; Origen = Bytes de nuestro sprite
ld      de,SPRTBL         ; Destino = Sprite Pattern Table
ld      bc,(NUMSPR*32)    ; Numero de bytes
call    LDIRVM            ; 05Ch - BIOS - Copy block to VRAM, from memory

; Esto lo realizamos para que se muestre de nuevo la pantalla
call    ENASCR            ; 044h - BIOS - Enable screen

ret                                           ; salir de la rutina INIT_GRAFICOS
;-----

;-----
; Tabla de los atributos de los sprites
tblATRsSPRs:
db      86,120, 0, 8      ; Coord.Y,Coord.X,nºSPR,Color
;-----

;-----
; Sprite Numero 1
; generated by TinySprite
SPR1:
; --- Slot 0
; color 15
DB $FF,$FF,$FE,$FC,$F0,$F0,$F0,$FE
DB $FE,$FE,$FE,$FE,$FE,$FE,$FE,$FF
DB $FF,$FF,$1F,$1F,$1F,$1F,$1F,$1F
DB $1F,$1F,$1F,$1F,$1F,$1F,$1F,$FF

```

Aquí finaliza el código de nuestro [HolaSPR1.asm](#) copia todo menos esta línea.

Ahora como siempre vamos a explicar el código, estoy seguro que a medida que avanzáis con los tutoriales ya sabéis lo que hace la mayor parte de este código. Así que solo describiré lo nuevo.

La cabecera de nuestro código como siempre nombre y fecha de nuestro proyecto. Direcciones etc.

Esta es la primera vez desde que escribo los tutoriales que uso la sección de **CONSTANTES**, el que sabe de programación seguro que sabe lo que es esto, al igual que una variable. Esto le dice a nuestro código que el nombre **NUM_SPR = 1** y **NUM_ATR = 1** después veremos porque lo usamos.

```

;-----
; CONSTANTES
;-----
NUM_SPR equ 1 ; Numero de SPRITES
NUM_ATR equ 1 ; Numero de Atributos de Sprite

```

Ahora en la sección donde defino las variables del MSX he agregado 2 nuevas variables si las miras en el fichero de [VariablesMSX.txt](#) del [pack-MSX](#) te dará más información, pero te lo resumo, cada registro del VDP es almacenado en las variables del sistema MSX desde [\\$F3E0](#) hasta [\\$F3E6](#) y el registro de estado del VDP en [\\$F3E7](#) después más adelante veremos porque las he incluido.

```

RG1SAV equ $F3E0 ; Content of VDP(1) register (R#1)
STATFL equ $F3E7 ; Content of VDP(8) status register (S#0)

```

Esto es una cosa que no es necesaria, pero es cómoda para cuando miras un fichero ROM por dentro para saber que versión es, ya que este texto se mostrara como ASCII al ver la ROM con un editor.

```

; Identificacion
db "HolaSPR1",1Ah

```

En nuestra querida rutina [INIT_MODE_SCx](#) he agregado 2 cosas nuevas, ya sabes que esta es la rutina encargada de seleccionar el modo de video SCREEN 2, ya en ultimo tutorial vimos que añadí que se desactive el sonido de las teclas y ahora vamos a ver 2 cosas nuevas que he agregado a esta rutina.

La 1ª creo que poco hay que comentar pero aquí va, el BASIC activa unas teclas de función esas que sacan al pulsar las teclas F1 a F12 las palabras clave en BASIC como COLOR, RUN , etc. etc. Pues esta llamada a la BIOS lo que hace es desactivar todas estas teclas de función.

```

; Desactivar las teclas de funcion
call ERAFNK ; 0CCh - BIOS - Erase function key display

```

La 2ª y la más importante para la zona del tutorial donde estamos es la selección del modo de SPRs que vamos a utilizar y que te explique en la teoría, SPR de 8x8 o 16x16 normales o ampliados.

```

ld bc,$E201 ; B=E2 en binario es 11100010b y C=01 que es el registro del VDP
call WRTVDP ; 047h - BIOS - Write to any VDP register

```

Esto es lo único que tendrías que hacer para seleccionar SPRs de 16x16 no ampliados. Pero esto no puede quedar sin explicación y por eso en el código he añadido otras formas de hacerlo, incluso la explicación que ahora mismo voy a detallarte del porque de las cosas.

No sé si lo sabéis o no, pero el VDP es el procesador grafico encargado de manejar la VRAM, toda la información grafica y los sprites del MSX, este VDP el TMS9918 tiene 7 tiene Registros y uno de ESTADO para realizar el acceso a este procesador. No me voy a enrollar que esto no es el libro rojo del MSX. Pero si hay cosas básicas que hay que saber, y una de ellas es el modo de los SPRs.

En el registro 1 del VDP se manejan estos bits.

```

; Modificar el Registro 1 del VDP ( Video Display Procesor )
;
; bit 7 : 10000000b: 1= siempre a 1 (4/16k VRAM)
; bit 6 : 01000000b: 1= Pantalla activada | 0= Pantalla desactivada
; bit 5 : 00100000b: 1= interrupciones activadas | 0= interrupciones desactivadas
; bit 4 : 00010000b: 1= modo texto | 0= otros modos
; bit 3 : 00001000b: 1= modo multicolor | 0= otros modos
; bit 2 : 00000000b: 0= siempre a 0
; bit 1 : 00000010b: 1= sprites de 16x16 | 0= sprites de 8x8
; bit 0 : 00000001b: 1= sprites *2 Ampliados | 0= sprites *1 no ampliados

```

Cuando llamamos al la BIOS a la rutina [DISSCR](#) realmente lo que hace esta rutina es modificar el [BIT 6](#) del registro del VDP para desactivar la pantalla. Pero a nosotros en este registro lo que nos interesa son los [BITS 0 y 1](#) si te fijas en la información técnica extraída del RED BOOK del MSX que he puesto a modo de comentarios, si el [BIT 1](#) del [Registro 1](#) del VDP los ponemos a 1 selecciona SPRs de 16x16 y si lo ponemos a 0 selecciona SPRs de 8x8, si modificamos el [BIT 0](#) a 1 los SPRs serán ampliados y si lo ponemos a 0 serán normales. Puedes hacer 2 cosas o leer el registro 1 del VDP y modificar solo los [BITS](#) que nos interesa, o bien meter a saco el valor, siempre teniendo en cuenta los valores de los 8 bits del registro 1, según la tabla de bits que te he puesto a modo de comentarios.

Veamos de donde sale el valor [\\$E201](#) que ponemos en [ld bc,\\$E201](#) como BC es un registro de 16 Bits lo separamos y tenemos el Registro [C=01](#) y el Registro [B=E2](#) y llamamos a la rutina en BIOS [WRTVDP](#) a esta rutina se la llama con el número del registro del VDP que vamos a modificar en [C](#) y el valor a

escribir en el registro en **B** y llamamos con **call** WRTVDP y esta escribirá el valor **E2** en el **registro 1** del VDP.(**Seguro que te preguntas que es el valor E2**) abre la calculadora de Windows y en el menú ver selecciona el formato Programador. Ahora convierte el valor E2 de **hexadecimal** a **binario** y viendo los 0 y los 1 sabrás con la información de arriba de los **BITs** que hace este valor.
E2 en binario es **11100010** lo ponemos en vez de izquierda a derecha de arriba a abajo para explicarlo.

- 1-Bit 7 – Siempre a 1
- 1-Bit 6 – Pantalla activada
- 1-Bit 5 – Interrupciones activadas
- 0-Bit 4 – Como es Screen 2 OTRO MODO
- 0-Bit 3 – Como es Screen 2 OTRO MODO
- 0-Bit 2 – Siempre a 0
- 1-Bit 1 – Sprites de 16x16
- 0-Bit 0 – No ampliados

Por este valor nuestros sprites se mostraran en pantalla con el tamaño de 16x16 no ampliados.

Os recomiendo esta forma de cambiar el registro 1 del VDP igual que ponemos **ld bc,\$E201** se puede poner también de esta forma que os pongo debajo de este texto, donde puedes ver claramente los 8 bits y el ultimo valor el 1 que sería el registro a modificar en nuestro caso el registro 1. Para el Ejercicio1 del final del tutorial deberás activar esta opción en el código quitando el **;** en esta opción que te explico y poniendo el **;** antes del **ld bc,\$E201** para anular esta línea y dejarla como un comentario.

```
ld      bc, (11100010b<<8) | 1      ; BC=$E201 en binario
call    WRTVDP                      ; 047h - BIOS - Write to any VDP register
```

Con esto ya hemos terminado de explicar los cambios en la rutina INIT_MODE_SCx

Vamos con la rutina **INIT_GRAFICOS** otra vieja conocida que seguro que ya sabéis lo que hace, desactivamos la pantalla, borramos la NAMTBL, colocamos los gráficos en VRAM y mostramos de nuevo la pantalla, aquí usamos como siempre la llamada a la rutina en BIOS LDIRVM. para mover bytes de ROM/RAM a VRAM. Moviendo los bytes del SPR1 creado con el tiny a la Sprite Pattern Table.

```
; Colocar los bytes de los SPRs en la Sprite Pattern Table
ld      hl, SPR1                    ; Origen = Bytes de nuestro sprite
ld      de, SPRTBL                 ; Destino = Sprite Pattern Table
ld      bc, (NUM_SPR*32)           ; Numero de bytes
call    LDIRVM                    ; 05Ch - BIOS - Copy block to VRAM, from memory
```

Ya sabéis que cada SPR son 4 CHRs y como un CHR son 8 Bytes pues $4*8=32$ que son los bytes que ocupa un SPR como hemos definido una CONSTANTE esta **NUM_SPR=1** pues multiplicamos $1*32$ ahora que tenemos un solo SPR pero a medida que vayamos incorporando mas SPRs al código no tendremos que cambiar esta parte del código, solo poner los bytes de otros SPRs y modificar la variable NUM_SPR cambiando su valor por el numero de SPRs que vayamos agregando al código.
 Vamos con otra parte explicada en la teoría de este tutorial referente al apartado donde explico los Atributos de los Sprites (**ATRs de los SPRs**)

Esta es la tabla que hemos definimos en el código y que mostrara nuestro primer SPR.

```
;-----
; Tabla de los atributos de los sprites
ATRSPRs:
    db      86,120, 0, 8      ; Coord.Y,Coord.X,nºSPR,Color
;-----
```

Vamos a colocar un SPR en las **Coordenadas Y=86 y X=120** de la pantalla que es el centro. Usaremos el SPR1 que empieza en el **CHR 0** y le daremos el color **Rojo=8**

Después de esta tabla es donde tenéis que pegar los bytes en formato DB's que hemos dibujado con el programa TinySprite y que hemos exportado a formato ASM añadiéndole la etiqueta **SPR1**:

```
;-----
; Sprite Numero 1
; generated by TinySprite
SPR1:
; --- Slot 0
; color 15
DB $FF,$FF,$FE,$FC,$F0,$F0,$F0,$FE
DB $FE,$FE,$FE,$FE,$FE,$FE,$FE,$FF
DB $FF,$FF,$1F,$1F,$1F,$1F,$1F,$1F
DB $1F,$1F,$1F,$1F,$1F,$1F,$1F,$FF
```

Llegados a este punto en el que tenemos los CHRs de nuestro SPR en la SPRTBL, solo nos falta que ese SPR se muestre en la pantalla, por eso he agregado en el bucle principal del Ejemplo este código.

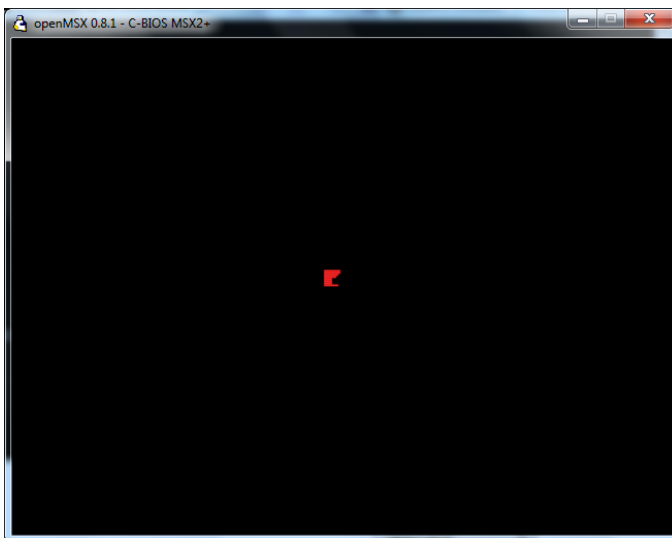
```
; Mostrar nuestro primer sprite
; Mover los ATRs de los SPRs en ROM/RAM a la SPRTBL en VRAM
ld    hl,tblATRSPRs ; Tabla de los ATRs de los SPRs en ROM
ld    de,SPRATR      ; Destino la Sprite Attribute Table
ld    bc,(NUM_ATR*4) ; Numero de Bytes
call  LDIRVM         ; 05Ch - BIOS - Copy block to VRAM, from memory
```

Esto leerá los bytes de la tabla **tblATRSPRs** y los colocara en la VRAM en la SPRATR. Ya sabes que que cada ATR de SPR son 4 valores como hemos definido una CONSTANTE esta **NUM_ATR=1** pues multiplicamos 1*4 ahora que tenemos un solo ATR pero a medida que vayamos incorporando mas ATRs al código no tendremos que cambiar esta parte del código, solo poner las tablas de otros ATRs y modificar la variable NUM_ATR cambiando su valor por el numero de ATRs que vayamos agregando al código. Este será el SPR1 en el Plano 0, ya que es nuestro primer SPR en esta zona de la VRAM.

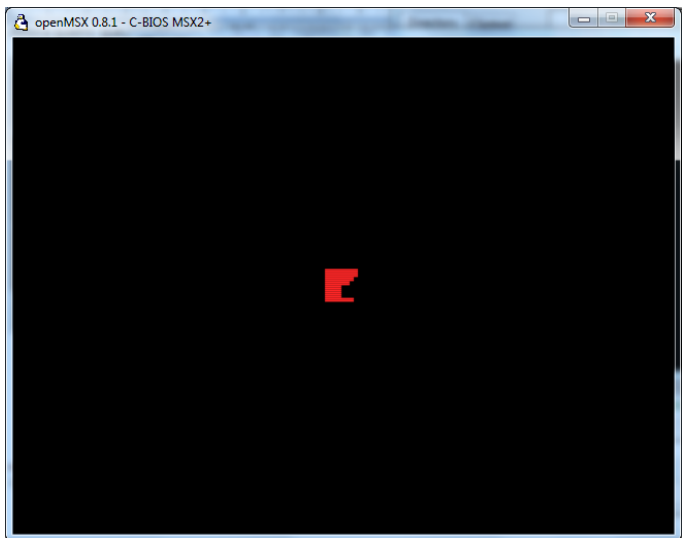
Ya estáis preparado para compilar y ejecutar la ROM de nuestro nuevo ejemplo el resultado tiene que ser la primera imagen de este tutorial.

EJERCICIO 1:

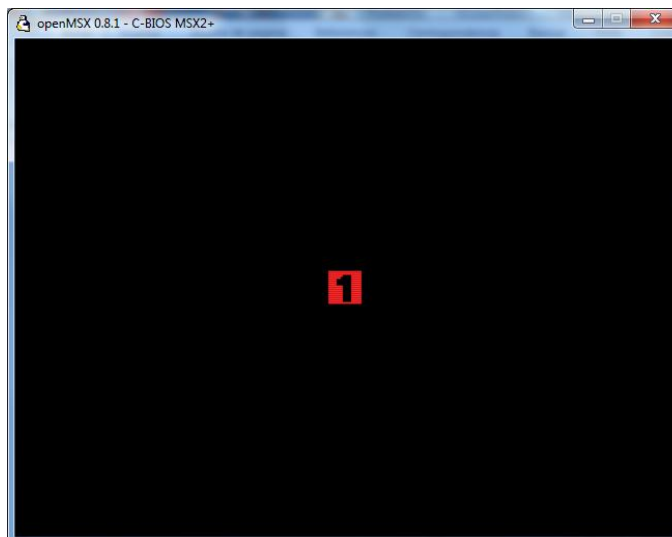
Ahora quiero que modifiquéis los BITS del registro 1 del VDP para que veas cómo seleccionar los diferentes modos de los SPRs compila y ejecuta cada cambio en la ROM de este ejemplo para que veas el resultado que se produce te adjunto las 4 capturas con los 4 formatos.



Sprites de 8x8 pixeles no ampliados



Sprites de 8x8 pixeles Ampliados



Sprites de 16x16 pixeles no ampliados



Sprites de 16x16 pixeles Ampliados

EJERCICIO 2:

Con lo que habéis aprendido podéis poner 2 SPRs en la pantalla usando el mismo grafico.

En la sección de **CONSTANTES** modificáis el valor 1 por un 2 en **NUM_ATR** ya que vamos a añadir un ATR nuevo sin añadir otro grafico, de esta manera no habrá que modificar código.

```
NUM_ATR      equ      2      ; Numero de Atributos de Sprites
```

Ahora añadir una segunda tabla de ATRs para nuestro SPR2

```
-----  
; Tabla de los atributos de los sprites  
tblATRSPRs:  
    db      86,120,  0,  8      ; Coord.Y,Coord.X,nºSPR,Color  
    db      86,144,  0,  3      ; Coord.Y,Coord.X,nºSPR,Color  
-----
```



Podéis ver en la segunda tabla, que añado la misma Coordenada Y, pero en la Coordenada X le sumo 24 para que se vea a la derecha del primero dejando 8 pixeles de separación entre el SPR1 y el SPR2 el numero de SPR lo dejo igual en 0 ya que solo tenemos 1 grafico, aunque ahora le pongo color Verde 3 al SPR. Compila y lanza la ROM el resultado será la imagen pequeña que ves encima de este texto.

Ahora vamos a ver sobre la pantalla del MSX lo que te explique en la teoría sobre los planos y la regla del 5º Sprite viéndolo con código y sobre una maquina real o emulador.

EJERCICIO 3:

Vamos con el orden de los planos, en el **EditPlus** vamos a **File – Save as...** Y el **HolaSPR1.asm** lo vamos a grabar como **HolaSPR2.asm** para añadirle las siguientes modificaciones.

En la sección **CONSTANTES** tiene que quedar así.

```
NUM_SPR      equ      5      ; Numero de SPRITES  
NUM_ATR      equ      5      ; Numero de Atributos de Sprite
```

Ahora en la tabla de ATRs de SPRs tiene que quedar como ves aquí.

```
-----  
; Tabla de los atributos de los sprites  
tblATRSPRs:  
    db      86,120,  0,  2      ; Coord.Y,Coord.X,nºSPR,Color  
    db      96,128,  4,  8      ; Coord.Y,Coord.X,nºSPR,Color  
    db      104,136,  8,  5      ; Coord.Y,Coord.X,nºSPR,Color  
    db      112,143, 12, 10      ; Coord.Y,Coord.X,nºSPR,Color  
    db      120,152, 16, 14      ; Coord.Y,Coord.X,nºSPR,Color  
-----
```

Ahora he creado mas SPRs con el tinysprite poniendo números del 1 al 5, Esto son sus bytes. Que teneis que sustituir por el del número 1 que tenemos ahora mismo en el código.

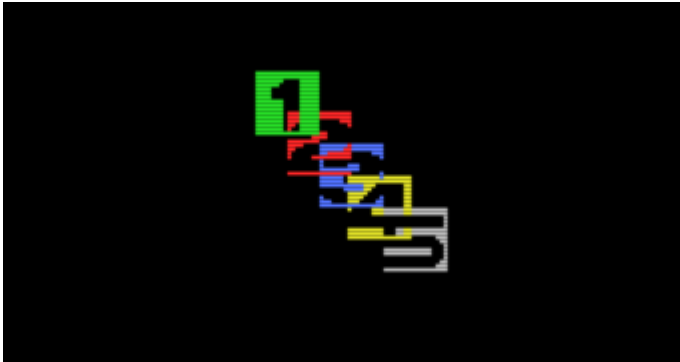
```
-----  
; Sprite Numeros 1 al 5  
; generated by TinySprite  
SPR1:  
; --- Slot 0  
; color 15  
DB $FF,$FF,$FE,$FC,$F0,$F0,$F0,$FE  
DB $FE,$FE,$FE,$FE,$FE,$FE,$FE,$FF  
DB $FF,$FF,$1F,$1F,$1F,$1F,$1F,$1F  
DB $1F,$1F,$1F,$1F,$1F,$1F,$1F,$FF  
SPR2:  
; --- Slot 0  
; color 15  
DB $FF,$FF,$F0,$C0,$80,$81,$03,$FF  
DB $F0,$C0,$80,$83,$00,$00,$00,$FF  
DB $FF,$FF,$07,$01,$00,$C0,$C0,$00  
DB $01,$03,$3F,$FF,$00,$00,$00,$FF  
SPR3:  
; --- Slot 1  
; color 15  
DB $FF,$FF,$F0,$C0,$80,$81,$FF,$FC  
DB $FC,$FC,$FF,$03,$00,$80,$E0,$FF  
DB $FF,$FF,$07,$01,$00,$C0,$C0,$01  
DB $01,$00,$E0,$E0,$00,$01,$03,$FF
```

```

SPR4:
; --- Slot 2
; color 15
DB $FF,$FF,$FE,$FC,$F8,$F0,$E0,$C1
DB $83,$03,$00,$00,$00,$FF,$FF,$FF
DB $FF,$FF,$03,$03,$03,$03,$83,$83
DB $83,$83,$00,$00,$00,$83,$83,$FF
SPR5:
; --- Slot 3
; color 15
DB $FF,$FF,$00,$00,$00,$1F,$1F,$00
DB $00,$00,$FF,$FF,$00,$00,$00,$FF
DB $FF,$FF,$01,$01,$01,$FF,$FF,$07
DB $03,$01,$F1,$F1,$01,$03,$07,$FF

```

Ahora que habéis incluido todo compila y lanza la ROM.



Aquí podéis ver que el SPR1 está en el plano 0 al ser el primer ATR y así sucesivamente.

Podéis ver con más claridad como cada SPR esta encima del otro SPR según el orden de los planos, que a su vez es el mismo orden en el que hemos ido situando cada ATR en la SPRATR.

EJERCICIO 4:

Vamos a ver otra característica del VDP al situar un SPR en la Coordenada Y 208, recuerda que te explique en la teoría de este tutorial que si pones un SPR en esa coordenada todos los SPRs del plano inferior a este dejaran de verse, vamos a comprobarlo. Esta es una forma de ocultar los SPRs en la pantalla en cualquier momento que nos interese realizar esta acción.

```

;-----
; Tabla de los atributos de los sprites
tblATRsSPRs:
    db      86,120,  0,  2    ; Coord.Y,Coord.X,nºSPR,Color
    db      96,128,  4,  8    ; Coord.Y,Coord.X,nºSPR,Color
    db     104,136,  8,  5    ; Coord.Y,Coord.X,nºSPR,Color
    db     112,143, 12, 10    ; Coord.Y,Coord.X,nºSPR,Color
    db     120,152, 16, 14    ; Coord.Y,Coord.X,nºSPR,Color
;-----

```

Modificáis la **coordenada Y** del 1º ATR y donde pone **86** cambiarlo por un **208**. Compila y lanza la ROM. El resultado es que no se verá ningún SPR en la pantalla. Vuelve a poner el 86 al ATR 1 y cambia el **96** del 2º ATR por un 208. Compila y lanza la ROM. Aquí puede ver que el SPR1 si se ve pero el resto no porque todos los planos inferiores al 1 no se ven cuando sitúas un SPR en la coordenada. Y 208. Si queréis probar con el 3º ATR ya es cosa vuestra.

EJERCICIO 5:

Vamos a ver qué pasa cuando ponemos 5 SPRs en la misma línea horizontal. Modificáis la tabla de ATR de esta manera y compila y lanza la ROM.

```

;-----
; Tabla de los atributos de los sprites
tblATRsSPRs:
    db      24, 24,  0,  2    ; Coord.Y,Coord.X,nºSPR,Color
    db      24, 48,  4,  8    ; Coord.Y,Coord.X,nºSPR,Color
    db      24, 72,  8,  5    ; Coord.Y,Coord.X,nºSPR,Color
    db      24, 96, 12, 10    ; Coord.Y,Coord.X,nºSPR,Color
    db      24,120, 16, 14    ; Coord.Y,Coord.X,nºSPR,Color
;-----

```



Como puedes ver en la imagen el 5º SPR está al lado del 4 pero no se ve.

EJERCICIO 6:

Vamos a mover 2 pixeles más abajo cada SPR para que no coincidan todos en la misma línea.

; Tabla de los atributos de los sprites

tblATRSPRs:

db	24,	24,	0,	2	; Coord.Y,Coord.X,nºSPR,Color
db	26,	48,	4,	8	; Coord.Y,Coord.X,nºSPR,Color
db	28,	72,	8,	5	; Coord.Y,Coord.X,nºSPR,Color
db	30,	96,	12,	10	; Coord.Y,Coord.X,nºSPR,Color
db	32,	120,	16,	14	; Coord.Y,Coord.X,nºSPR,Color



Una vez que modifiquéis la tabla de ATRs compila y lanza la ROM.

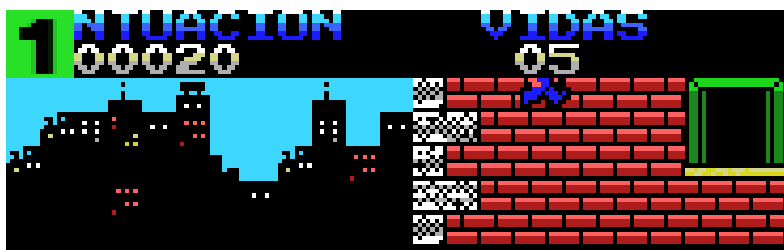
Como podéis ver ahora si se muestra parcialmente el SPR nº5. (Porque se muestra parcialmente?)

La respuesta está en que la norma del 5º SPR solo se cumple cuando hay 5 SPR en la misma línea horizontal. Fíjate en la imagen que te pongo a continuación.



Pongo 2 líneas imaginarias para que veas que la parte que desaparece del SPR5 es justo la que coincide con el SPR1 esto sucede porque esa parte del SPR5 es la única zona donde coinciden los 5 SPRs en la misma línea.

TRUCO en JumpinG. Una forma de sacar provecho a nuestro favor de la regla del 5º SPRs.



He puesto el SPR1 en la imagen para explicarte este truco

Te pongo una fragmento de mi juego JumpinG, si os fijáis cuando el prota salta y pasa por encima de los marcadores aquí puedes ver los pies, la cabeza no pasa por encima del número de vidas.

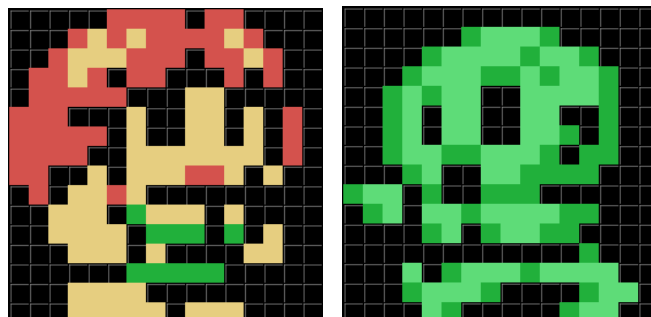
Realmente si lo está haciendo pero aquí se produce la regla del 5º SPR desapareciendo la parte del prota cuando coinciden 5 SPR en la misma línea, esto se consigue colocando 4 SPR con color transparente donde está el numero 1, estos SPRs no se ven porque su color es transparente y están en los planos 0,1,2,3 después sitúo el prota en el plano 5, así cuando el prota salta por encima del marcador al haber 5 SPRs en la misma línea, desaparece esta parte del prota dejando ver el marcador.

TRUCO del Zombie Incident para salvar la regla del 5º Sprite.



Fijaros en estas imágenes del juego Zombie Incident

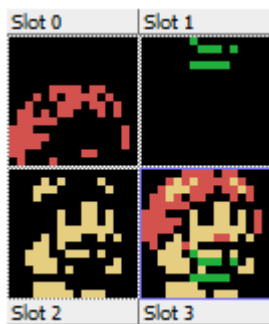
La Prota y el Zombie Verde de la imagen
Os pongo un Zoom de los 2 para la explicación.



La prota tiene 3 colores que son 3 SPRs mas 2 colores del Zombie verde otros 2 SPRs esto quiere decir que aparentemente hay 5 SPR en la misma línea. Como es esto posible? Haciendo Flickering? NO Se notaría el parpadeo, el truco está en que en la misma línea horizontal no hay más de 4 SPRs. Seguro que os estáis preguntando... pues no lo entiendo si hay 5 colores hay 5 SPRs.

Os desvelo el secreto en la siguiente página.

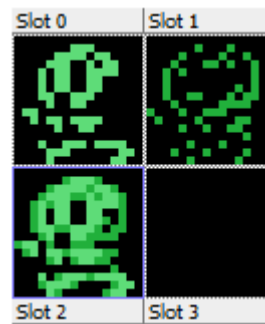
Estos son los 3 SPRs de la Prota.



SPR1 - El pelo Rojo
SPR2 - El vestidito verde
SPR3 - El cuerpo de la Prota

El resultado mezclado de los 3.

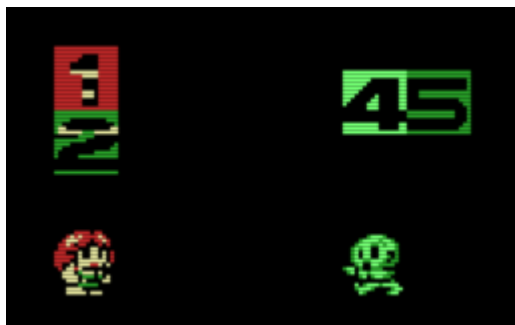
Y estos los 2 SPRs del Zombie Verde



SPR1 - 1º color
SPR2 - 2º color

La mezcla de los 2

Si os fijáis en la posición del pelo y del vestido verde no coinciden con la posición donde están en la mezcla, esto es así porque el SPR del pelo va situado justo encima del SPR del vestido y el cuerpo de la prota en medio de los 2, vamos a verlo con el ejemplo de los SPRs con números.



Aquí podéis ver que el SPR1-el Pelo Rojo está situado justo encima del SPR2 que es el vestido Verde, estos 2 SPRs nunca están en la misma línea horizontal mientras que el cuerpo que es el SPR3 está entre medias de los dos. Y aunque en la imagen veas que el SPR4 y SPR5 están separados realmente están juntos pero es para que lo veas bien que hay 5 SPRs, aquí si puedes apreciar que en la misma línea horizontal solo hay 4 SPRs a la vez ya que el SPR1 y el SPR2 no se mezclan en la misma línea horizontal.

Ahora tirar 2 líneas imaginarias en el SPR1 en esa línea horizontal solo coinciden los SPR 1,3,4,5 y si vuelves a tirar 2 líneas imaginarias en el SPR2 en esa línea horizontal solo coinciden los SPRs 2,3,4,5 en cualquiera de los 2 casos siempre hay un máximo de 4 SPRs en la misma línea. Por eso no hay que colocar otro SPR enemigo en la misma línea horizontal porque entre la prota y los 2 enemigos sumarian 6 SPRs en la misma línea.



Pues esto es todo para la primera parte del mundo de los Sprites.

Espero ver vuestros comentarios y vuestras capturas de pantallas, en los BLOG´s o FORO´s del tutorial.

Espero que haya sido de vuestro total agrado y nos vemos en el próximo tutorial. Donde veremos cómo podemos paliar el efecto del 5º SPR, además de incorporar movimientos a los SPRs con la lectura de los cursores y Joystick. Y veremos por primera vez como trabajar con la memoria RAM.

Link para el acceso a los ejemplos o código del tutorial.

<http://www.dimensionzgames.com/wp-content/uploads/downloads/2012/03/Tutorial6.rar>

José Vila Cuadrillero

"ES DETESTABLE ESA AVARICIA ESPIRITUAL QUE TIENEN, LOS QUE SABIENDO ALGO, NO PROCURAN LA TRANSMISION DE ESOS CONOCIMIENTOS."

Miguel de Unamuno

Escritor y Filósofo.

(Bilbao 1864 - Salamanca 1936)