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     0F3E9h  ; 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 0F3E9h “
Puedes ver todas las variables del sistema del MSX en el fichero de texto variablesMSX.txt en el fichero comprimido Pack-MSX.rar de la primera parte del tutorial.
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 06Ch 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 RETornamos 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      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     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 en código " 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 los ves en Entry.. y los parámetros de salida los ves en Exit.. además de los registros del z80 modificados 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      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     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 de memoria donde está el primer carácter del 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 en la cabecera de la rutina y describas todo 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, 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 contaras 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
; LA COLUMNA Y FILA ES LEIDA CON EL TEXTO
; Entrada:              HL = Dir. de la cadena de texto a pintar
; Salida:               ninguna
; Reg. 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          ; pasa 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
;-------------------------------------------------------
; db   Columna,Fila,"Cadena de texto",0 = Fin de texto
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.

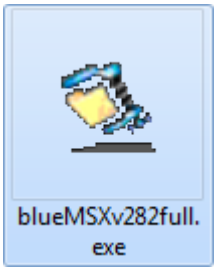
Si utilizamos etiquetas condicionales en el asMSX como @@bucle: podremos emplear este nombre en otra rutina que tenga un bucle, si ponemos bucle: solo lo podremos emplear una vez en esa rutina.

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)

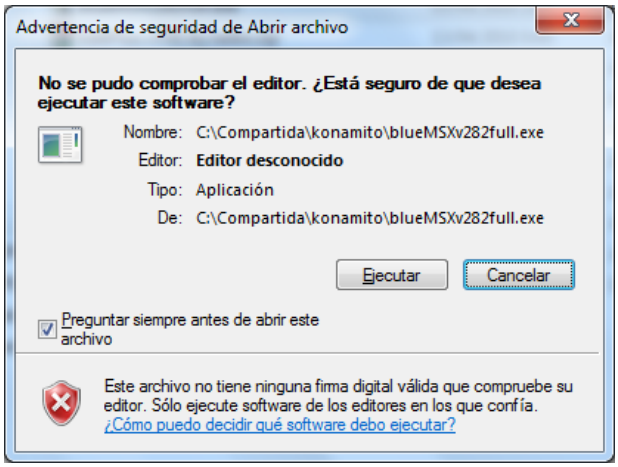
Para fijar un punto de Interrupción en nuestro programa hemos de incluir la directiva .BREAK en el punto del código donde queremos que el Debugger del blueMSX pare la ejecución de nuestro programa y podamos seguir el código paso a paso.

```
;-------------------------------------------------------
INICIO:
; INICIO DEL PROGRAMA
;-------------------------------------------------------
        .break                 ; directiva del asMSX para crear un punto de interrupción aquí mismo.
        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
```

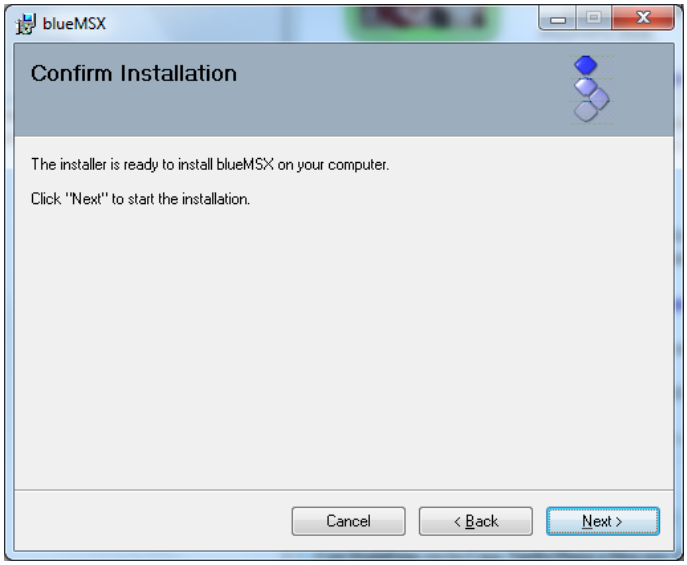
Como puedes ver he puesto el punto de interrupción en el bucle principal del programa o donde la directiva .START tiene el punto de entrada al programa ya que quiero tracear mi programa desde el principio pero la directiva .break podemos ponerla en cualquier parte del código que queramos.
En el PACK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

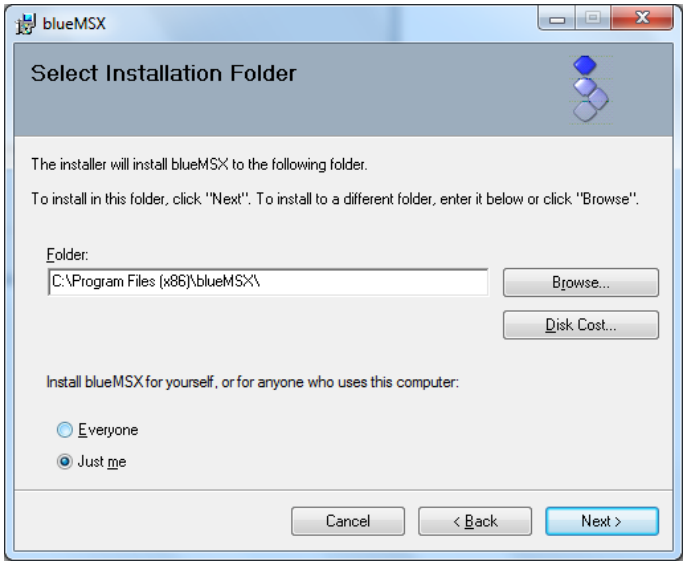


Pulsa doble clic sobre el icono del blueMSXv282full.exe para que comience la Instalación.

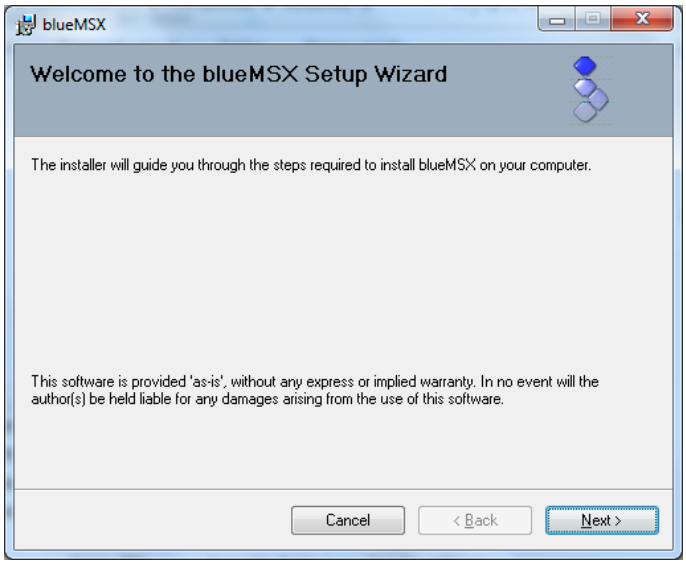


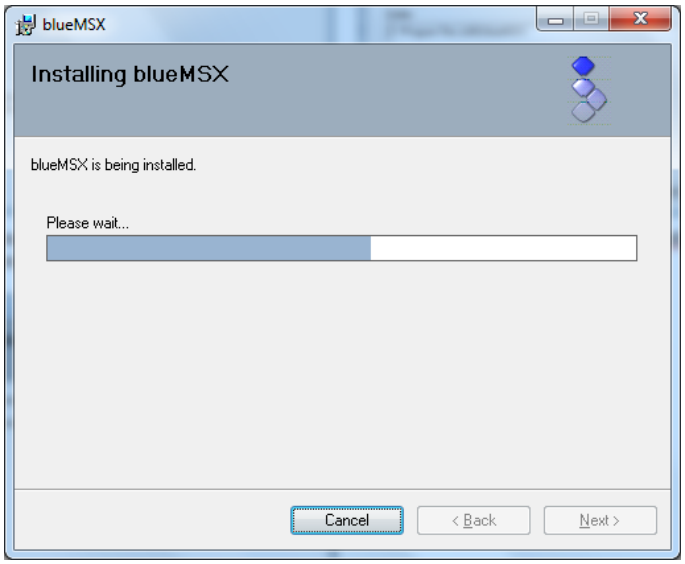
Pulsa el botón Ejecutar.



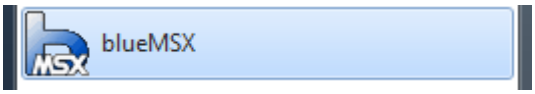


Comienza la instalación Pulsamos Next – Seleccionamos el directorio donde instalar el BlueMSX yo dejo el directorio por defecto que me sugiere el programa.



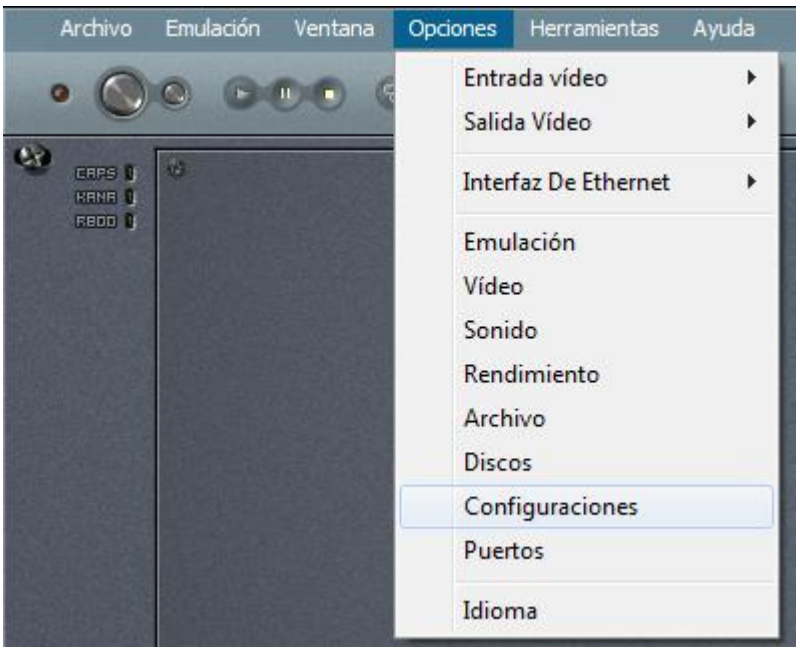


Pulsamos el Botón Next – y esperamos a que se instale el BlueMSX y en la siguiente ventana en Close.

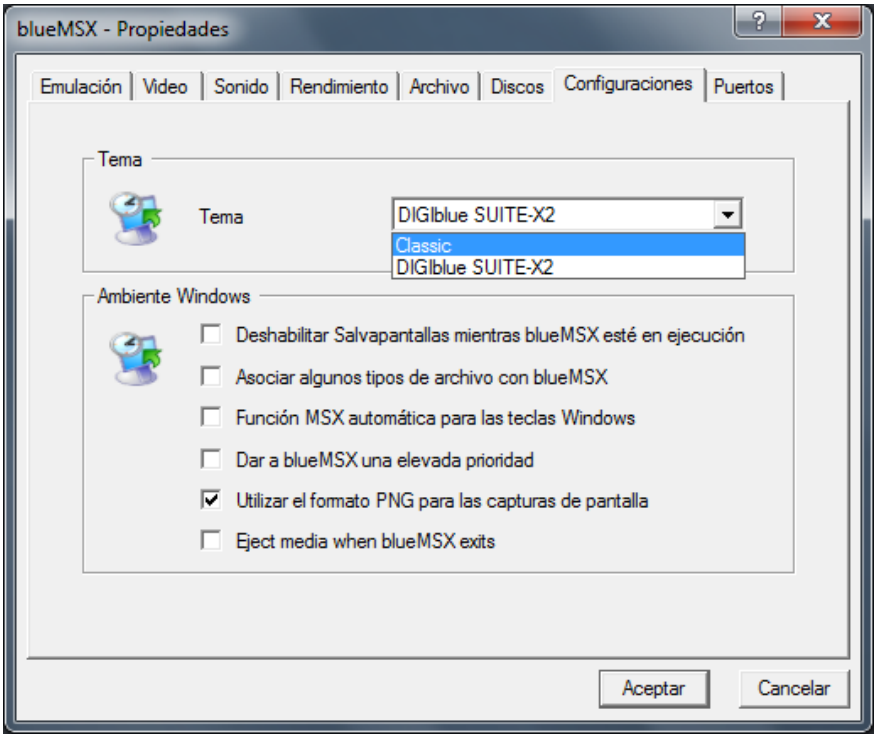


Lanzamos el programa una vez finalizada la instalación.

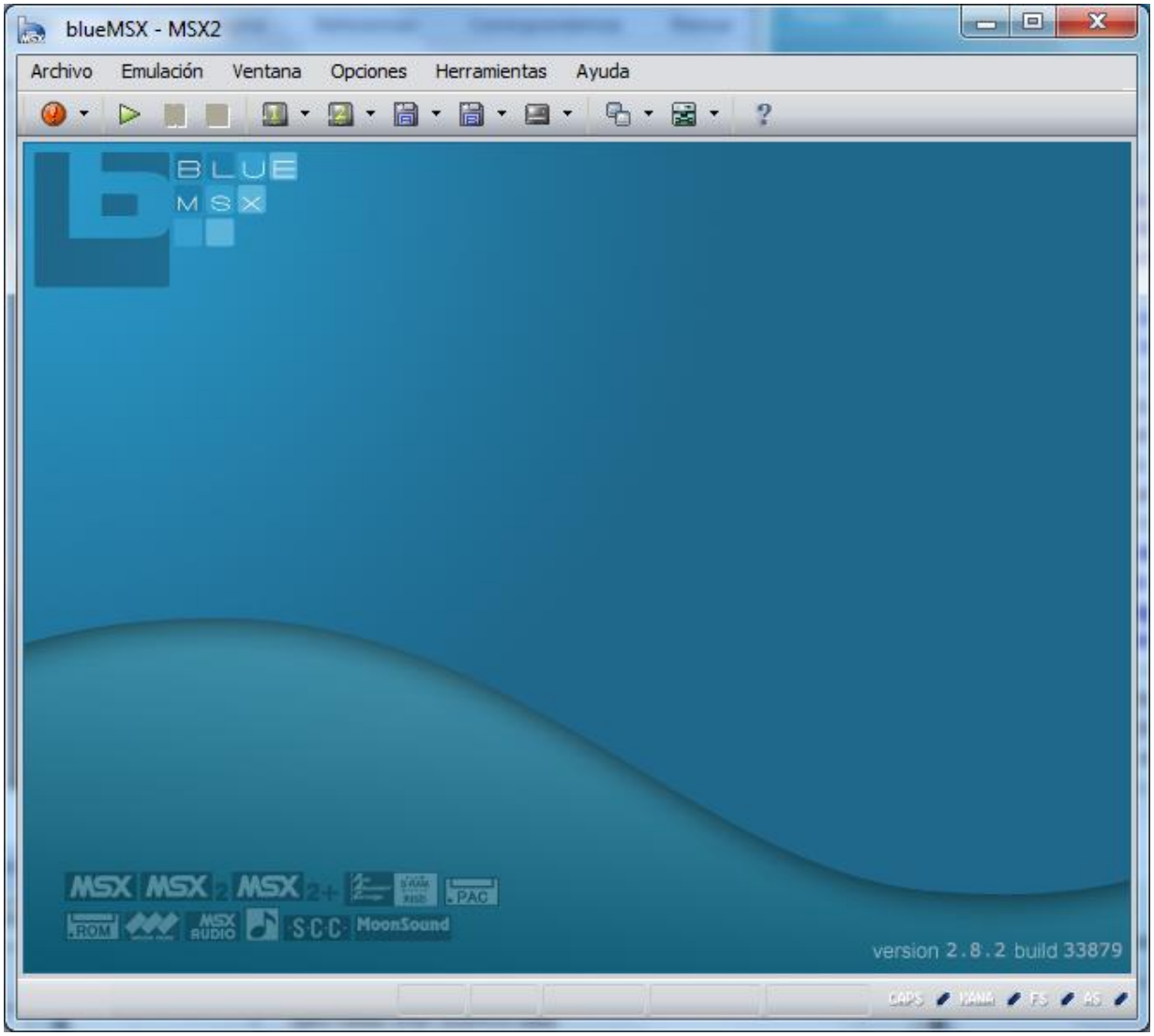
Aquí tenemos el BlueMSX con su SKIN bonito pero vamos a cambiarlo por el clásico que nos viene mejor para trabajar con el tutorial.



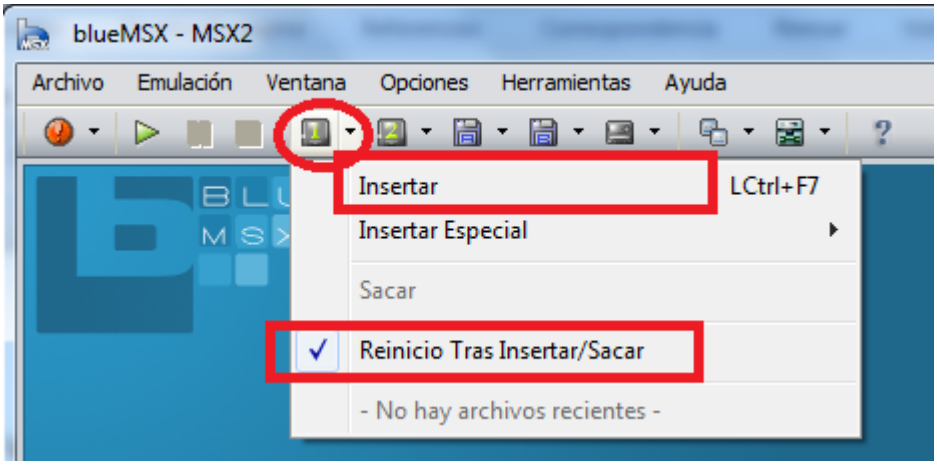
Pulsa en los menús de arriba del BlueMSX en Opciones y selecciona Configuraciones.



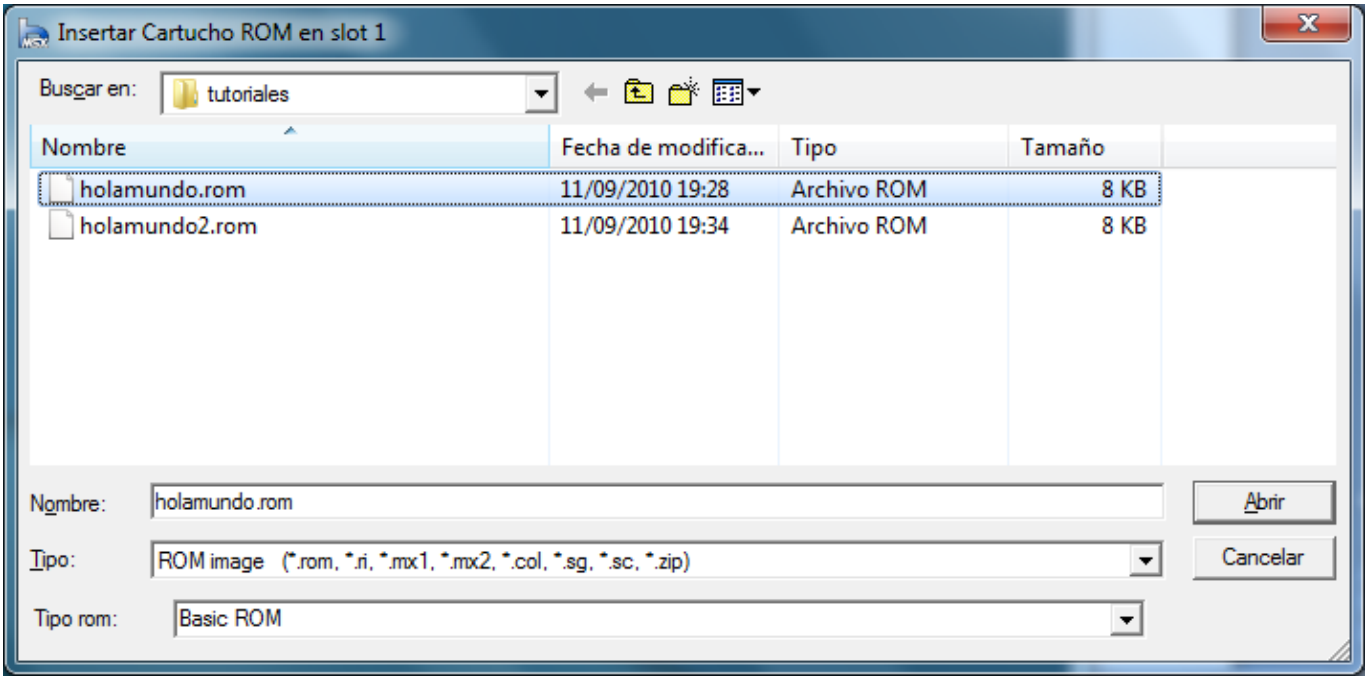
En la ventana que se abre esta que ves a la izquierda de este texto modifica el Tema pasando de DIGIblue-SUITE-X2
a Classic y pulsa el botón Aceptar.



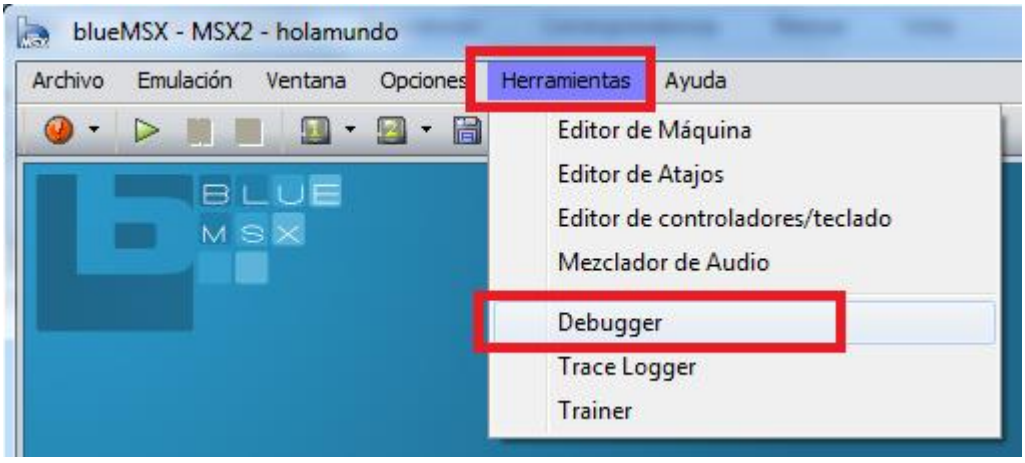
Esta va a ser la imagen que tendrá nuestro BlueMSX si has realizado bien los pasos.
Vamos a cargar nuestra ROM “HolaMundo“ en el blueMSX



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.

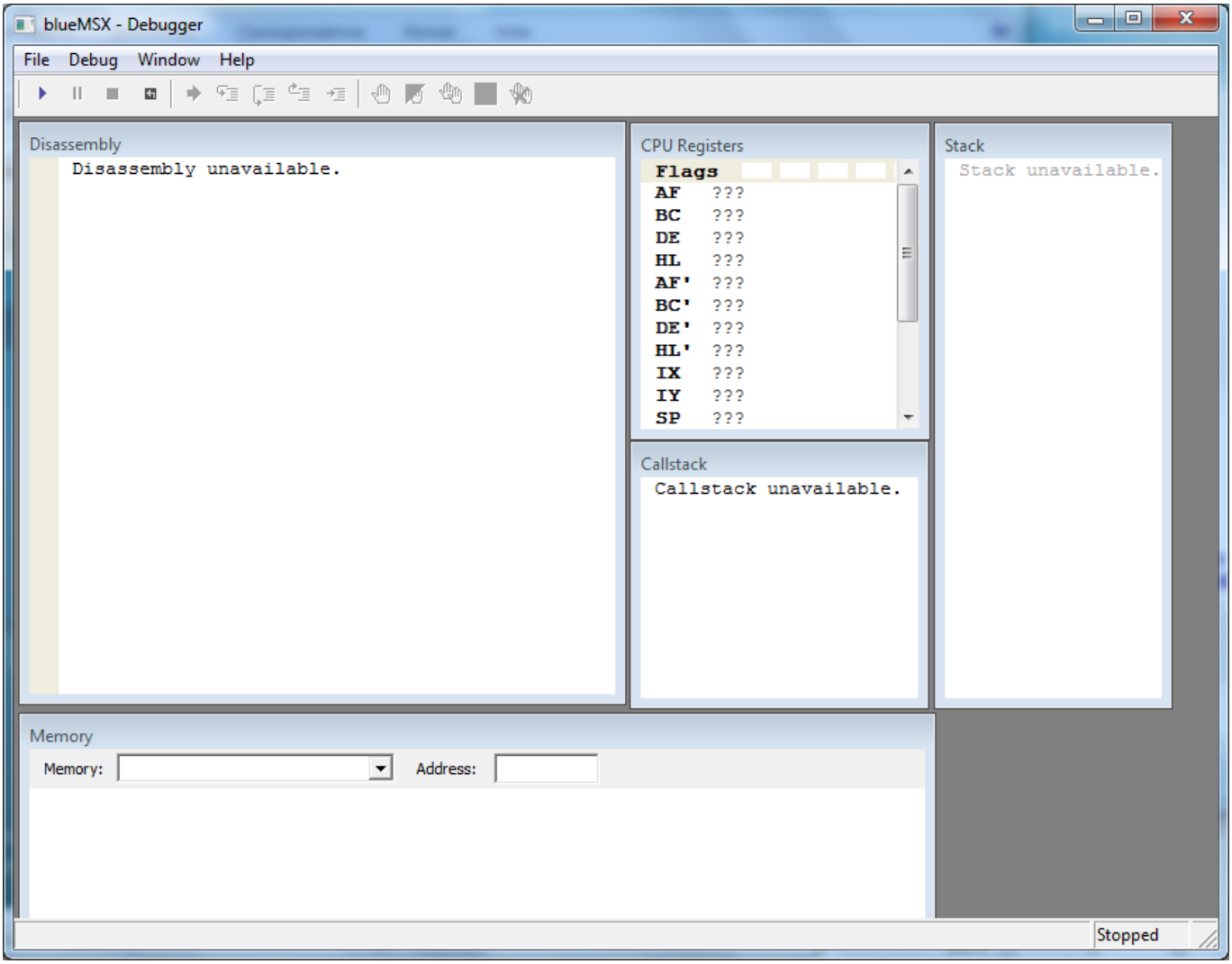


Selecciona la ROM en mi caso HolaMundo.rom tu cárgala desde donde la hayas guardado y con el nombre que le hayas puesto a la ROM.

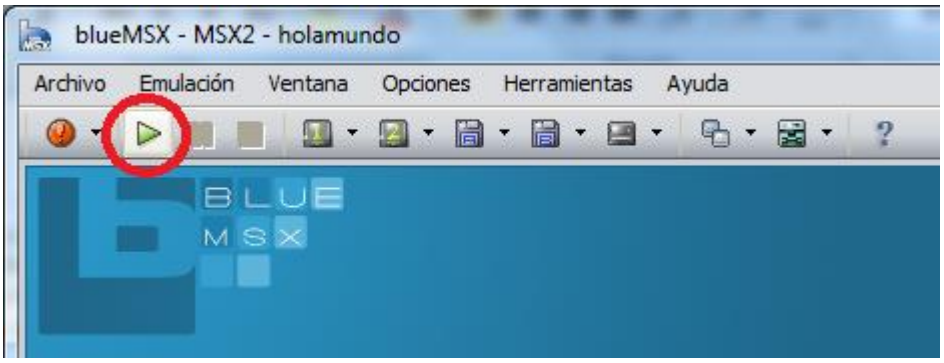


Ahora que tenemos cargada nuestra ROM Antes de ponerla en marcha vamos a abrir el DEBUGGER

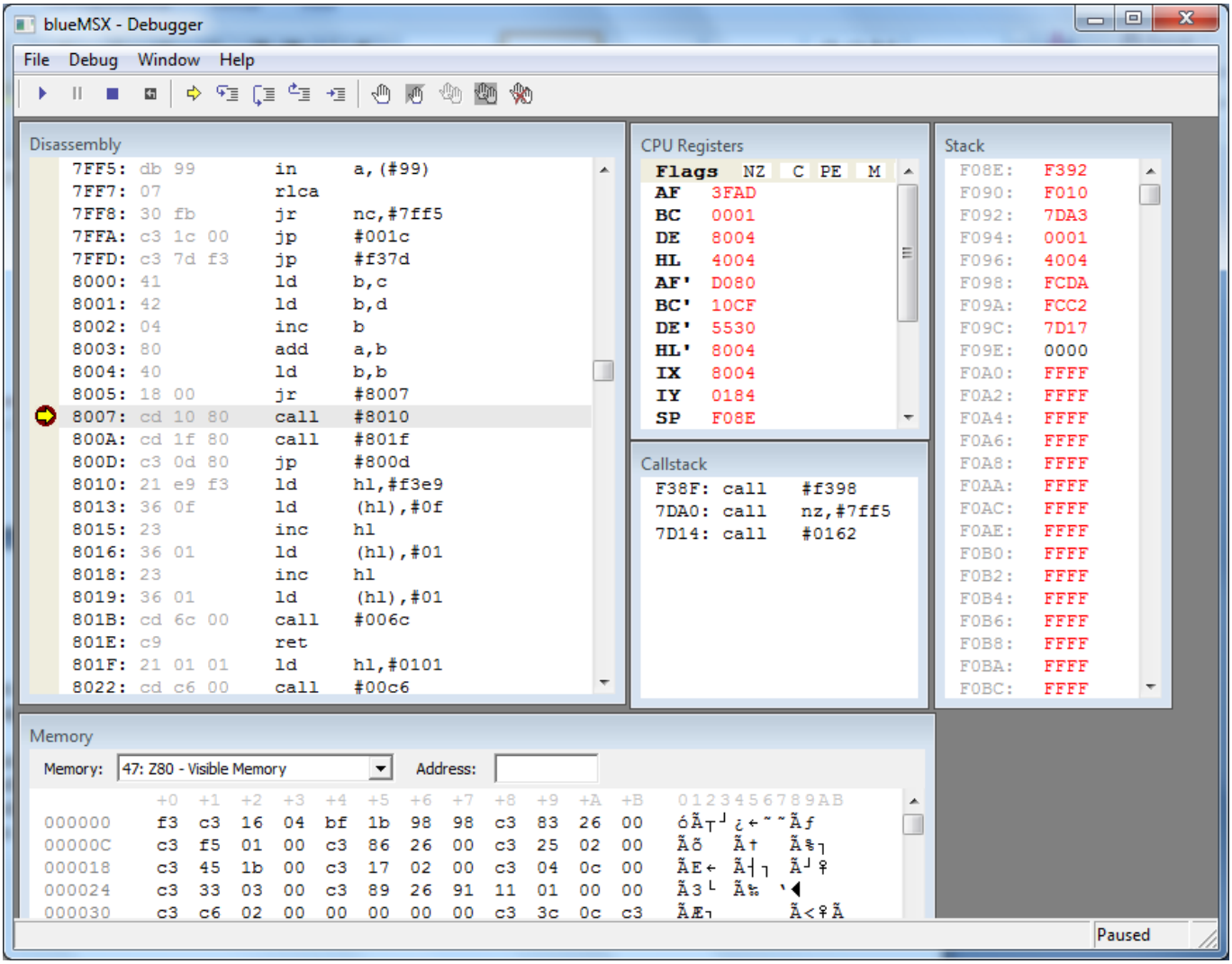
Pulsa en los menús de arriba en Herramientas y pulsa en la opción Debugger.



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 saltara al código en el punto donde hemos puesto la directiva del asMSX .break para que paralice la ejecución del programa en ese punto.

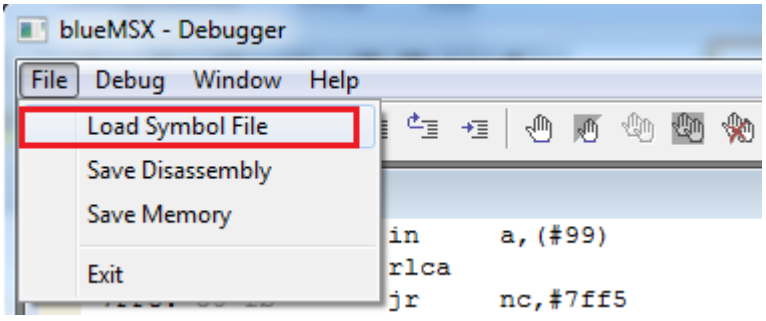


Pulsa el botón del Play en el BlueMSX y vamos a poner en marcha nuestra ROM veras que la ROM no se ejecuta y nos muestra directamente el código desensamblado en el DEBUGGER.

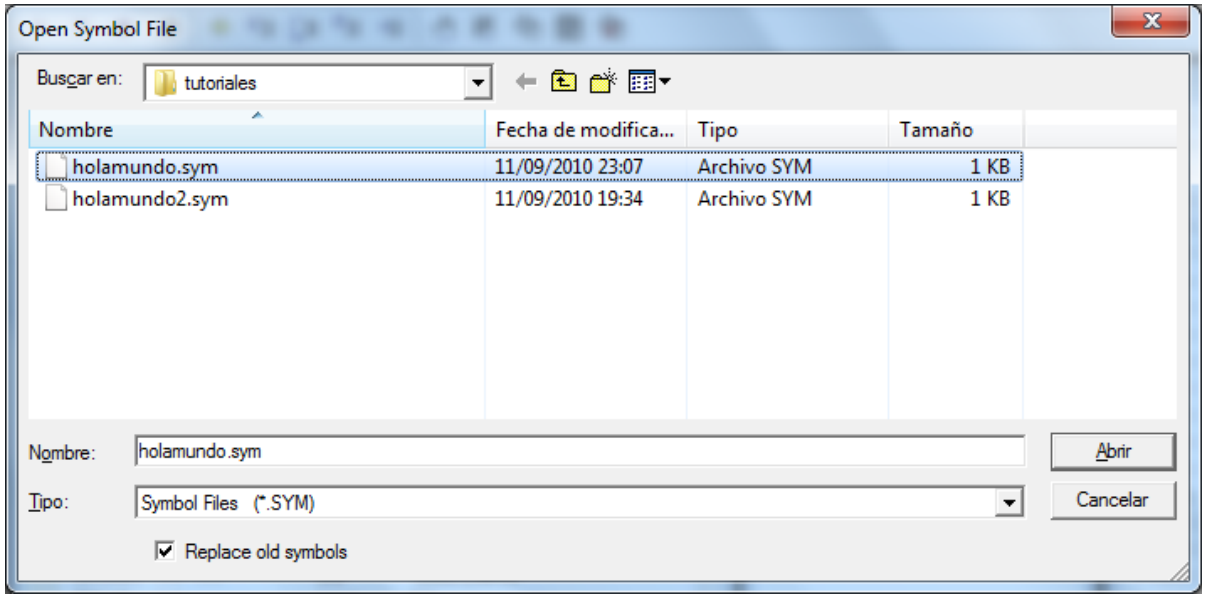


Ya tenemos nuestro programa parado en el punto que nosotros le hemos dicho. Pero qué pasa con el código no aparecen los nombres de las rutinas o de las variables. (Tranquilo esto es normal)

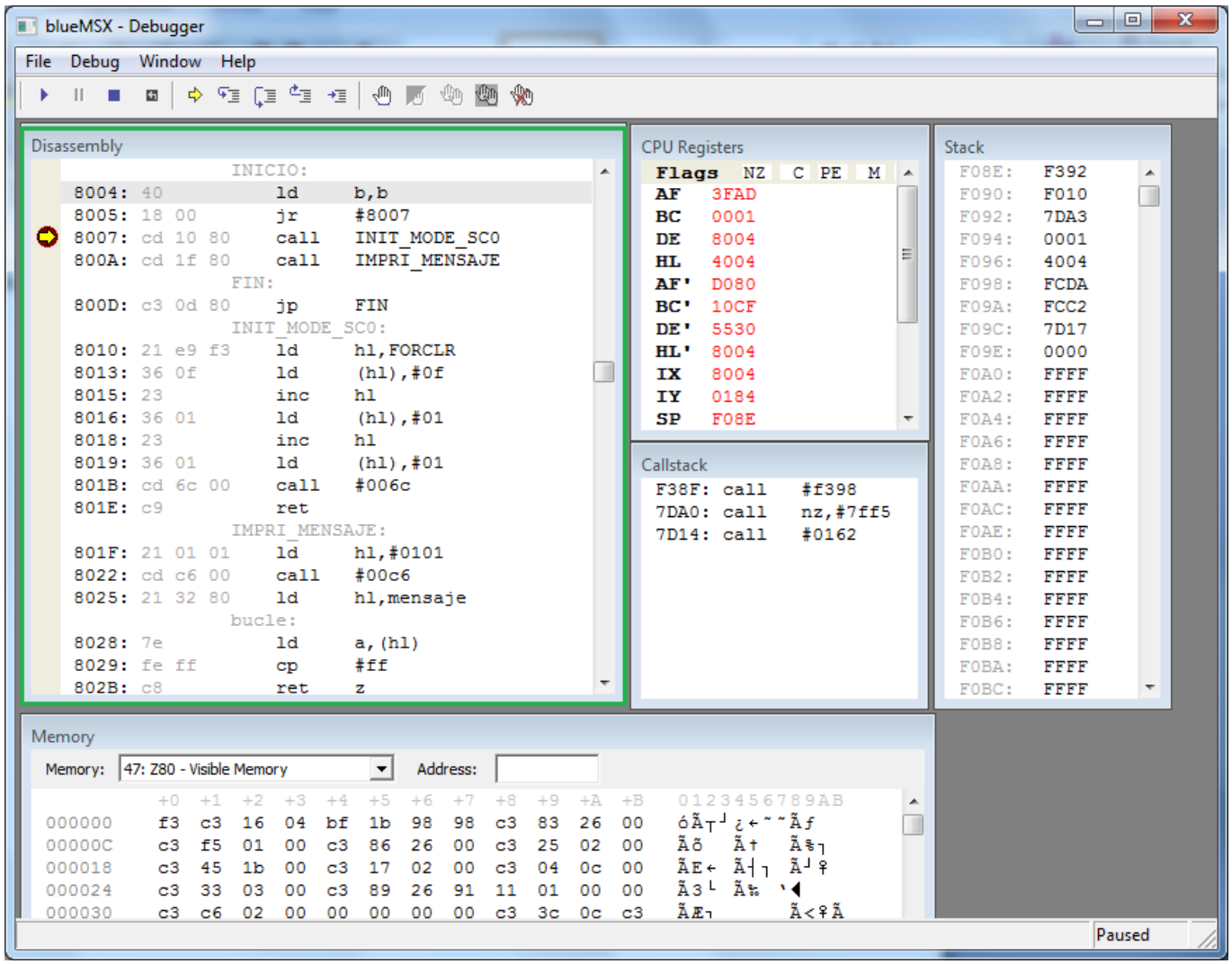
Lo que pasa es que realmente esta desensamblado el código y el no conoce los nombres ya que estos son sustituidos por direcciones de memoria a la hora de compilar, para ver el código de una manera más parecida al lenguaje ensamblador siendo nosotros los creadores del programa tenemos un fichero de símbolos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

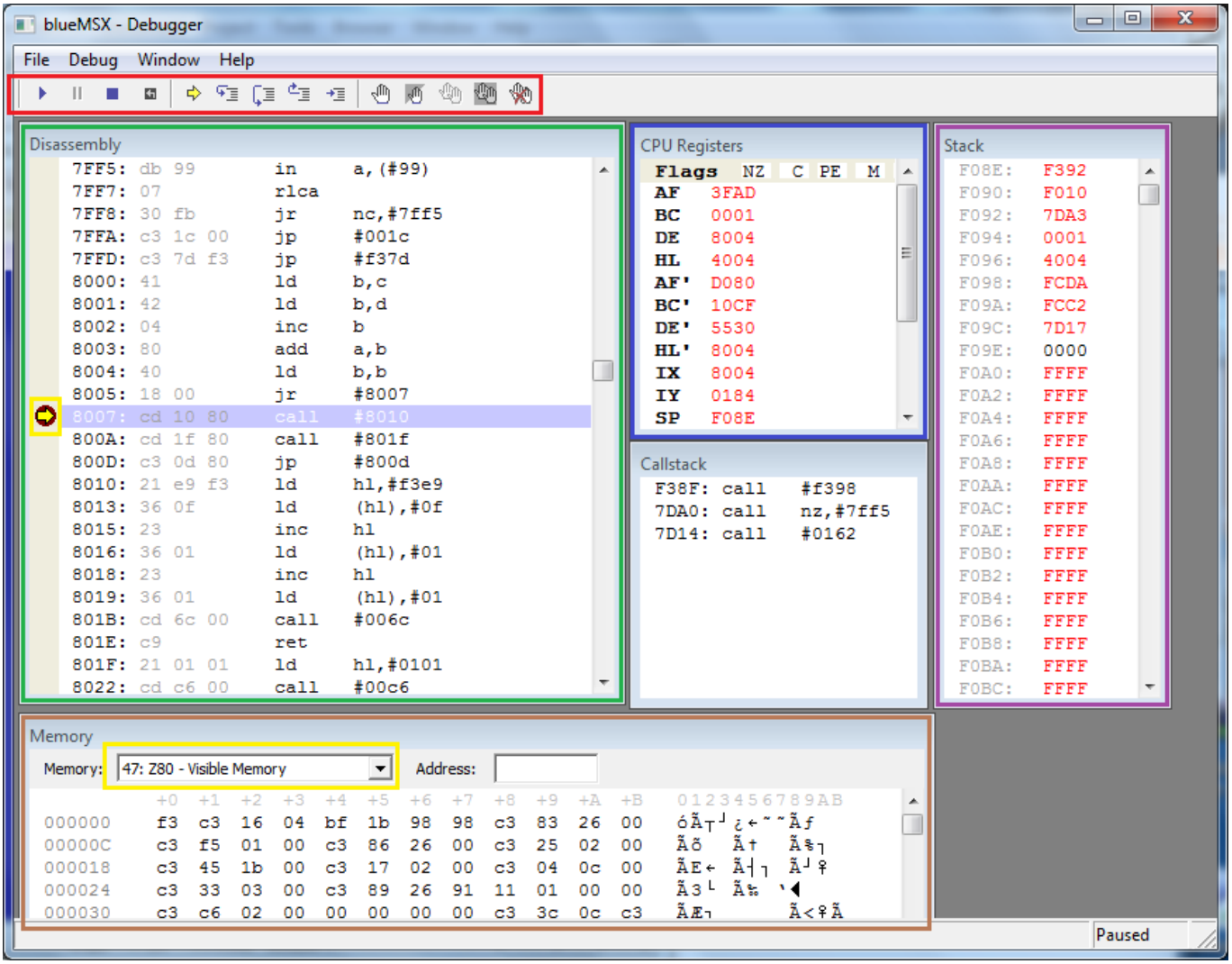


Como puedes ver en el recuadro en verde ahora ya salen los nombres de nuestras rutinas y variables si bien los valores salen en Hexadecimal, por eso yo prefiero trabajar en esta numeración ya que es la más empleada en código maquina aparte del binario claro. (Las llamadas a la BIOS no salen nombres.)

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 Stament: Cuando nos movemos en la ventana Dissasembly 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 Dissasembly 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 parara 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
F0A0:	FFFF
F0A2:	FFFF
F0A4:	FFFF
F0A6:	FFFF
F0A8:	FFFF
F0AA:	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.

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			

Recuerdas que en el código del primer programa tecleamos.

```
; Variables de sistema
      FORCLR  equ     0F3E9h  ; Foreground colour
```

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

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

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	0123456789AB
00F3E4	36	07	04	9f	f1	0f	04	04	c3	00	00	c3	6•┘ŸñЖ┘┘Ã Ã
00F3F0	00	00	0f	59	f9	ff	01	01	f0	fb	f0	fb	Ж Yùÿ ðûðû
00F3FC	53	5c	26	2d	0f	25	2d	0e	16	1f	53	5c	S\ -Ж%-♫┬ 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.

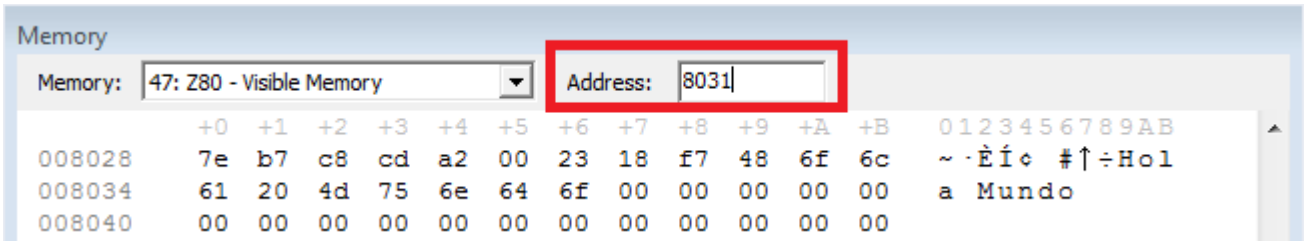
```
8007: cd 10 80     call    INIT_MODE_SC0
800A: cd 1f 80     call    IMPRI_MENSAJE
                FIN:
```

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.

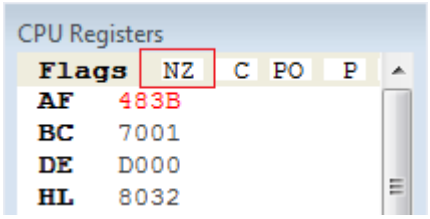
Como puedes ver arriba de este texto mensaje empieza en la posición 8031 de la memoria visible.



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 ascii en hexadecimal 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.



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 del 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.

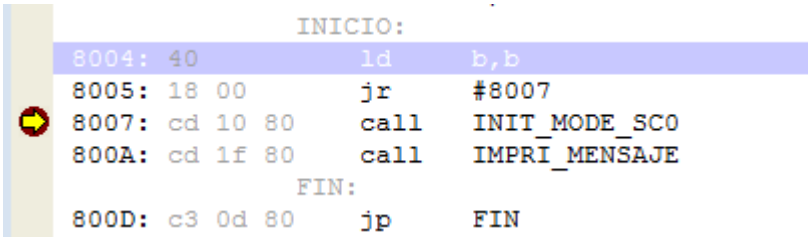
Cosas que debemos aprender del debugger para futuros usos no explicados anteriormente.

- **Los puntos de interrupción.**

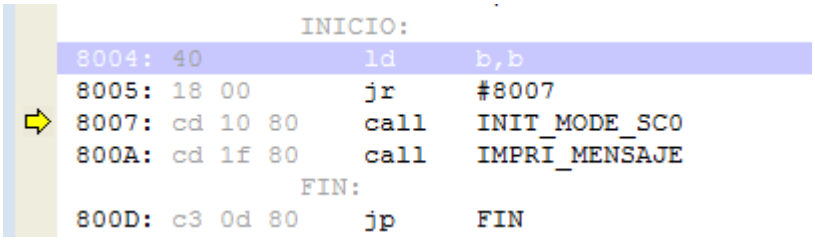
Cuando se desensambla el código de un programa o nuestro propio código podemos poner puntos de interrupción en cualquier parte de la memoria, esto se suele realizar para varios cometidos, pero te explico uno. Imagina que quieres ver cómo funciona una determinada rutina, pero quieres solo tracear esa 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.

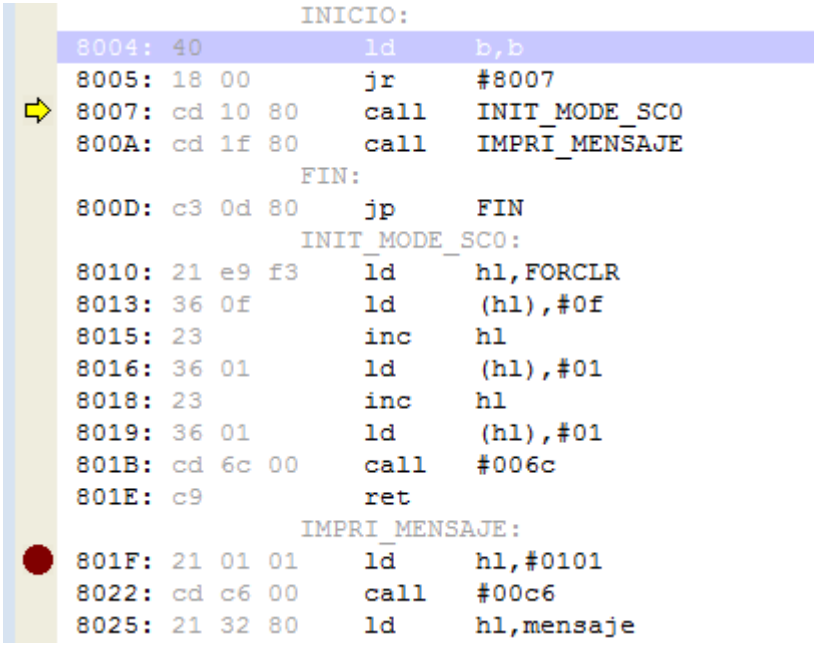
Si has cerrado todo ejecuta de nuevo el blueMSX lanza de nuevo el debugger, y carga el programa.



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.

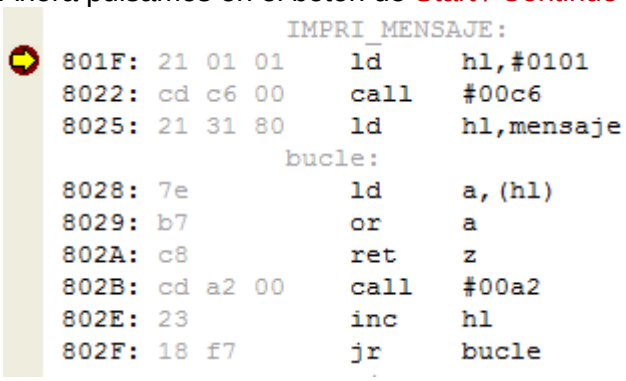


Como puedes ver en la imagen ya hemos quitado el punto de interrupción inicial, de igual manera que quitamos el punto de interrupción podemos ponerlo, siempre que pulsemos con el ratón en esa parte.



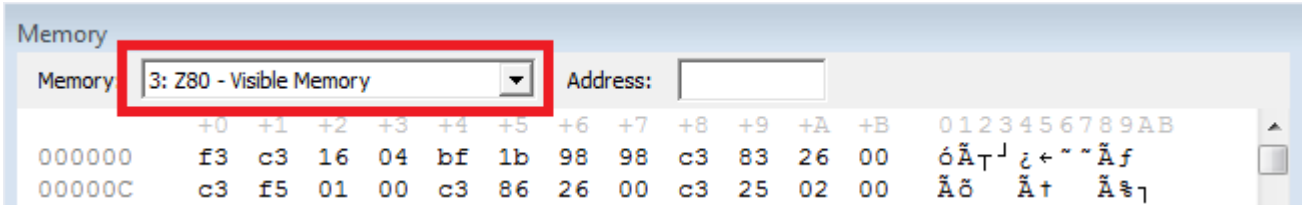
Ahora Fijamos el punto de interrupción justo donde empieza la rutina de impresión del mensaje, como puedes ver en la imagen encima de este texto.

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

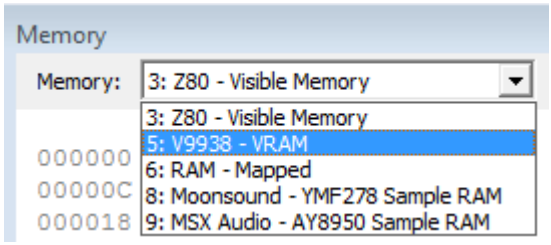


Como puedes ver se ha ejecutado todo el código anterior hasta detenerse donde hemos fijado el punto de interrupción.
Con esto damos por finalizado la sección sobre puntos de interrupción.

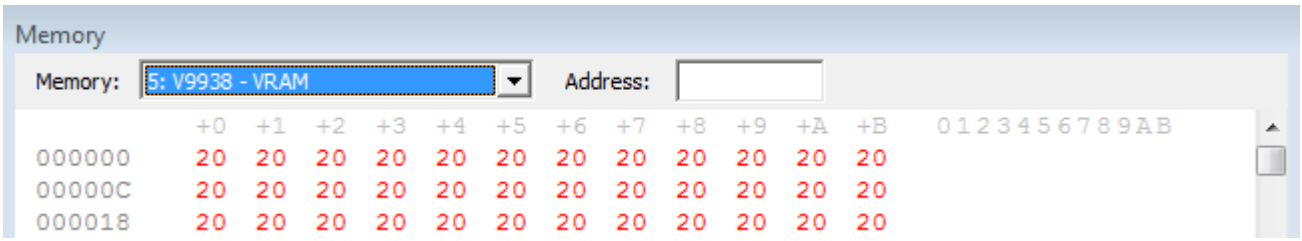
- **Manipulación y visualización de datos en la memoria.**



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.



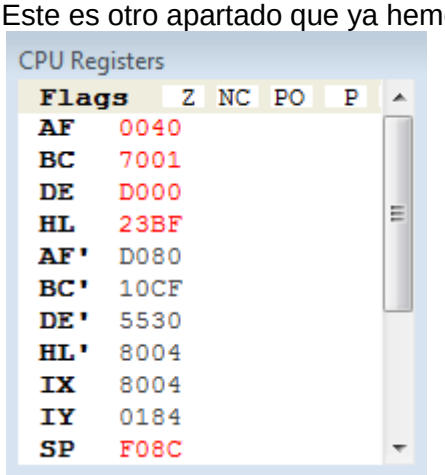
Pulsando en el flecha puedes ver la RAM z80 – Visible Mem
Selecciona v9938 – VRAM para ver los valores de la memoria de video
También se podría ver la memoria mapeada o los datos del chip de sonido.



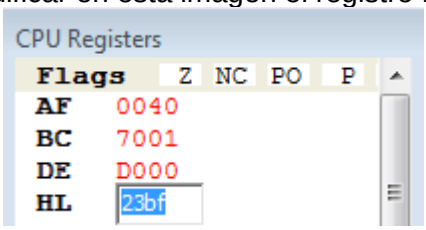
Como puedes ver en la imagen ya estaríamos viendo los valores hexadecimales de la VRAM.

- **Manipulación de FLAGS (Banderas) o Registros del Z80**

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



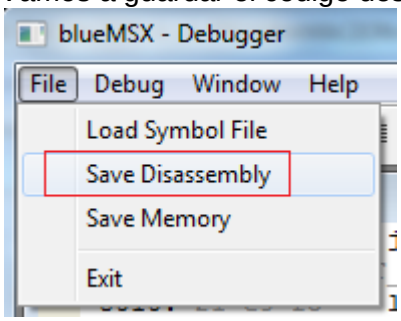
Al igual que has aprendido a modificar los Flags a nuestro antojo en determinadas ocasiones de nuestro traceo nos puede interesar modificar los valores de los registros, esto lo puedes realizar pulsando con el ratón sobre el valor que contiene el registro que queremos modificar en esta imagen el registro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



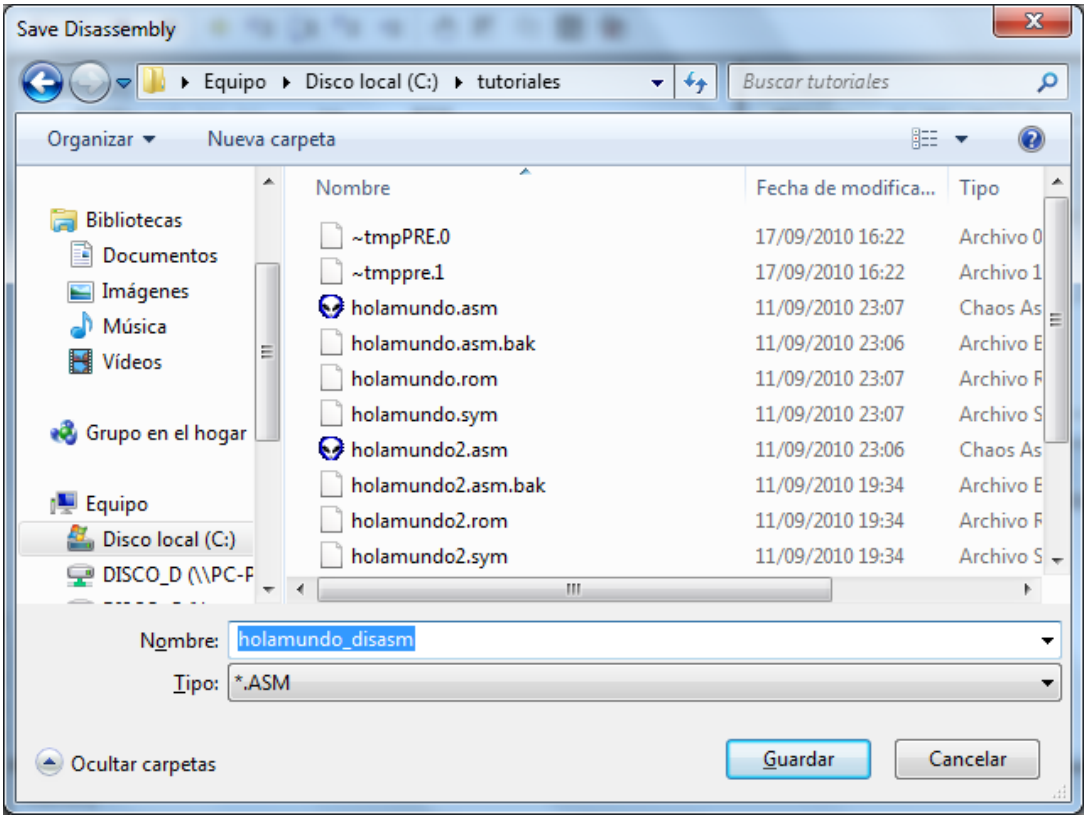
- **Volcado de la memoria y desensamblado**

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.

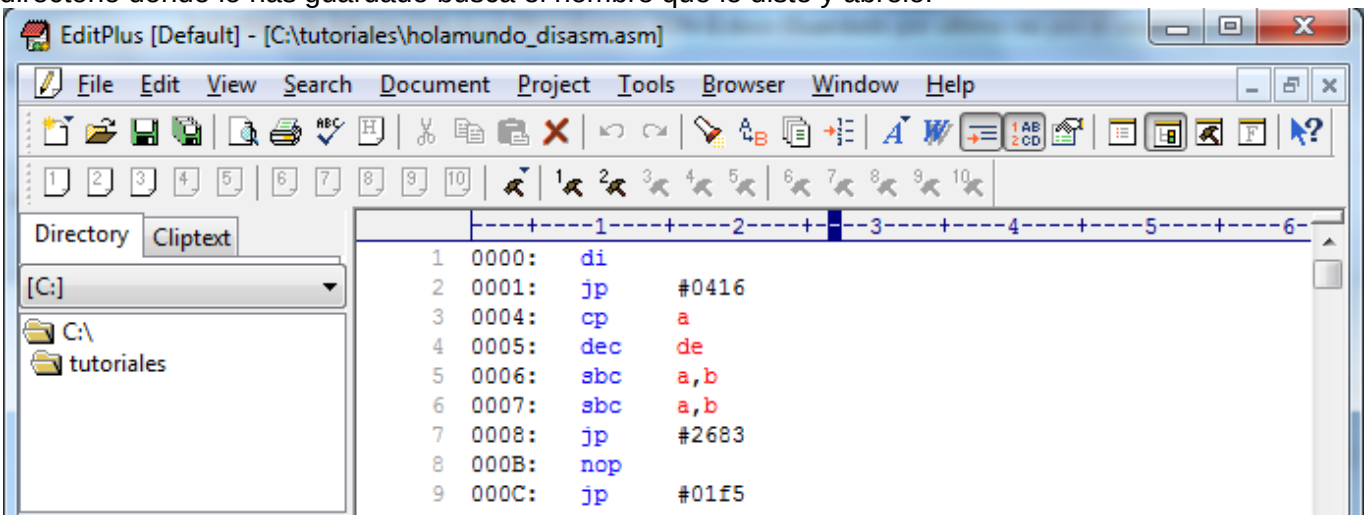


Pulsa en el menú File en el Debugger del blueMSX
Y selecciona la opción Save Disassembly

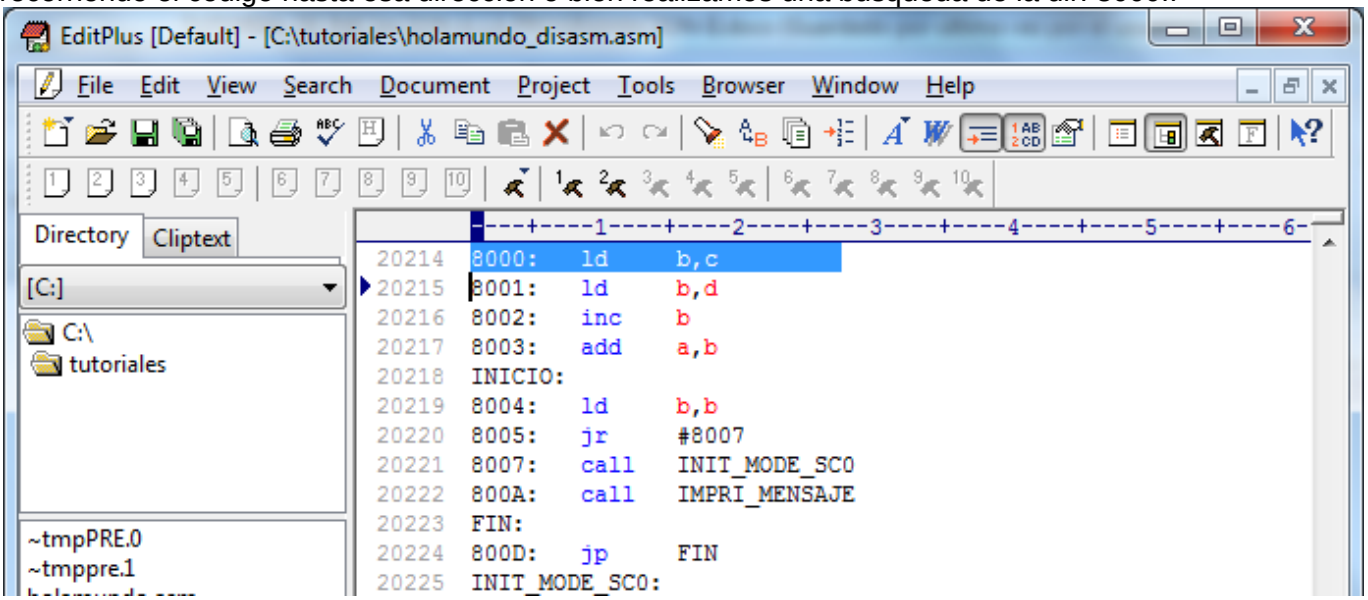


Dale un nombre al programa que queremos guardar el desensamblado y pulsa el botón Guardar. Deberían ponerle una opción al blueMSX para decirle desde y hasta que dirección de memoria quieres desensamblar ya que el blue lo hace desde la dirección de memoria 00000 hasta la 65535 todo vamos.

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.



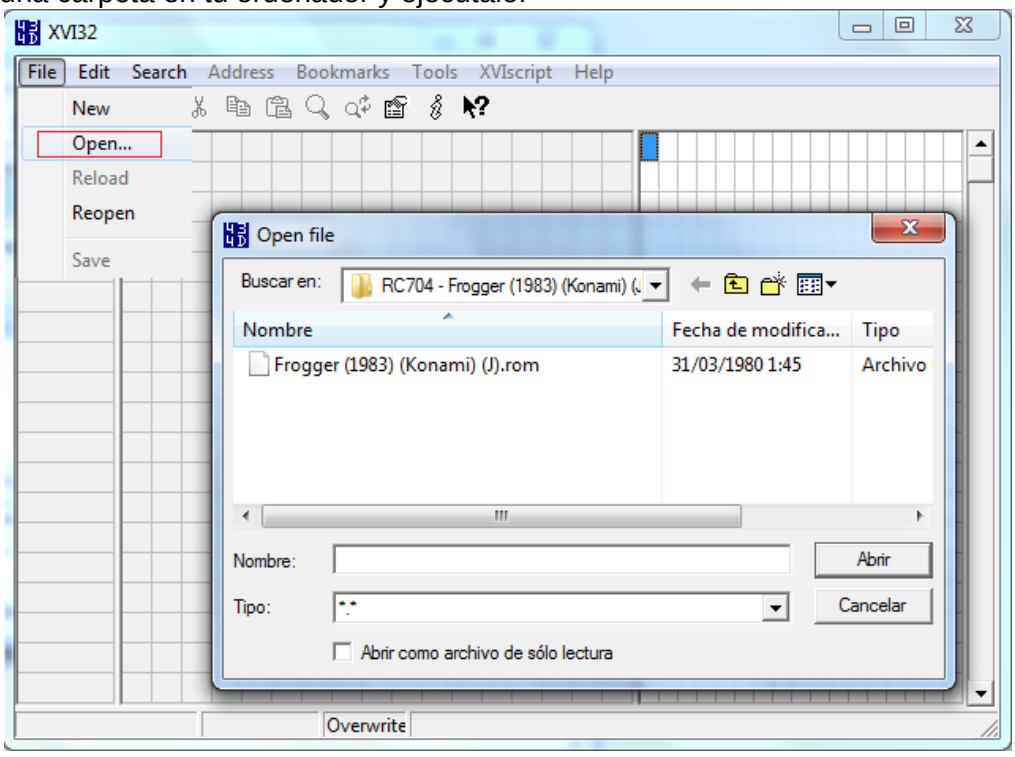
Aquí en la imagen de arriba puedes ver que el código desensamblado empieza en la dirección 0000 como nosotros sabemos que nuestro código empieza en la dirección 8000h Hexadecimal nos vamos recorriendo el código hasta esa dirección o bien realizamos una búsqueda de la dir. 8000:.



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 tracear 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.

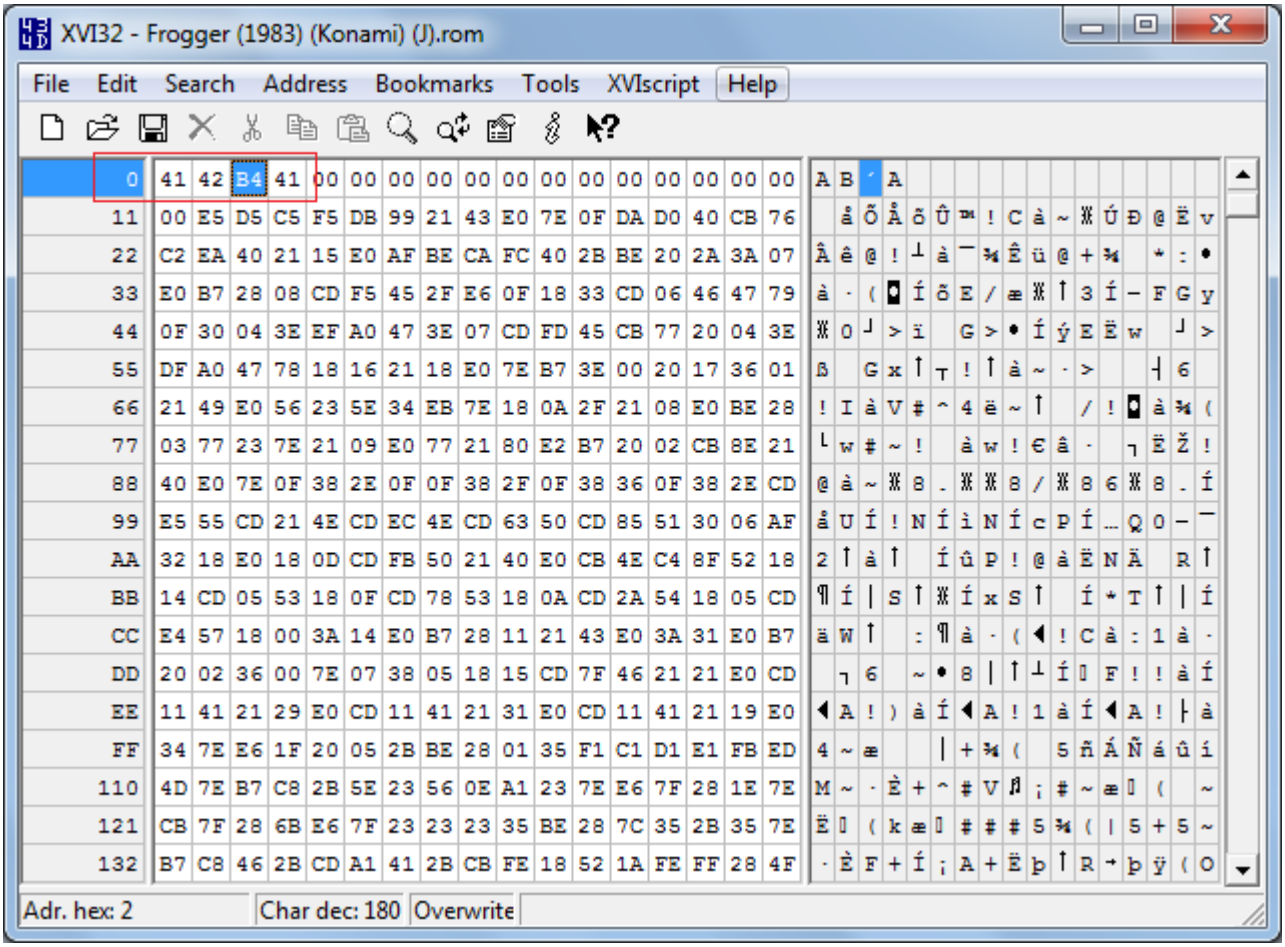
1 – Si no tienes ningún editor hexadecimal en tu ordenador puedes descargar este programa gratuito desde este enlace http://www.handshake.de/user/chmaas/delphi/download/xvi32.zip descomprímelo en una carpeta en tu ordenador y ejecútalo.



Vete al menú File y selecciona Open…

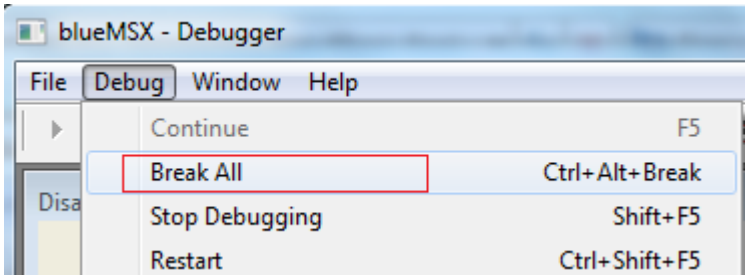
En el cuadro de dialogo Open file busca donde tengas bajada la ROM del Frogger selecciónala y dale al botón Abrir.

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

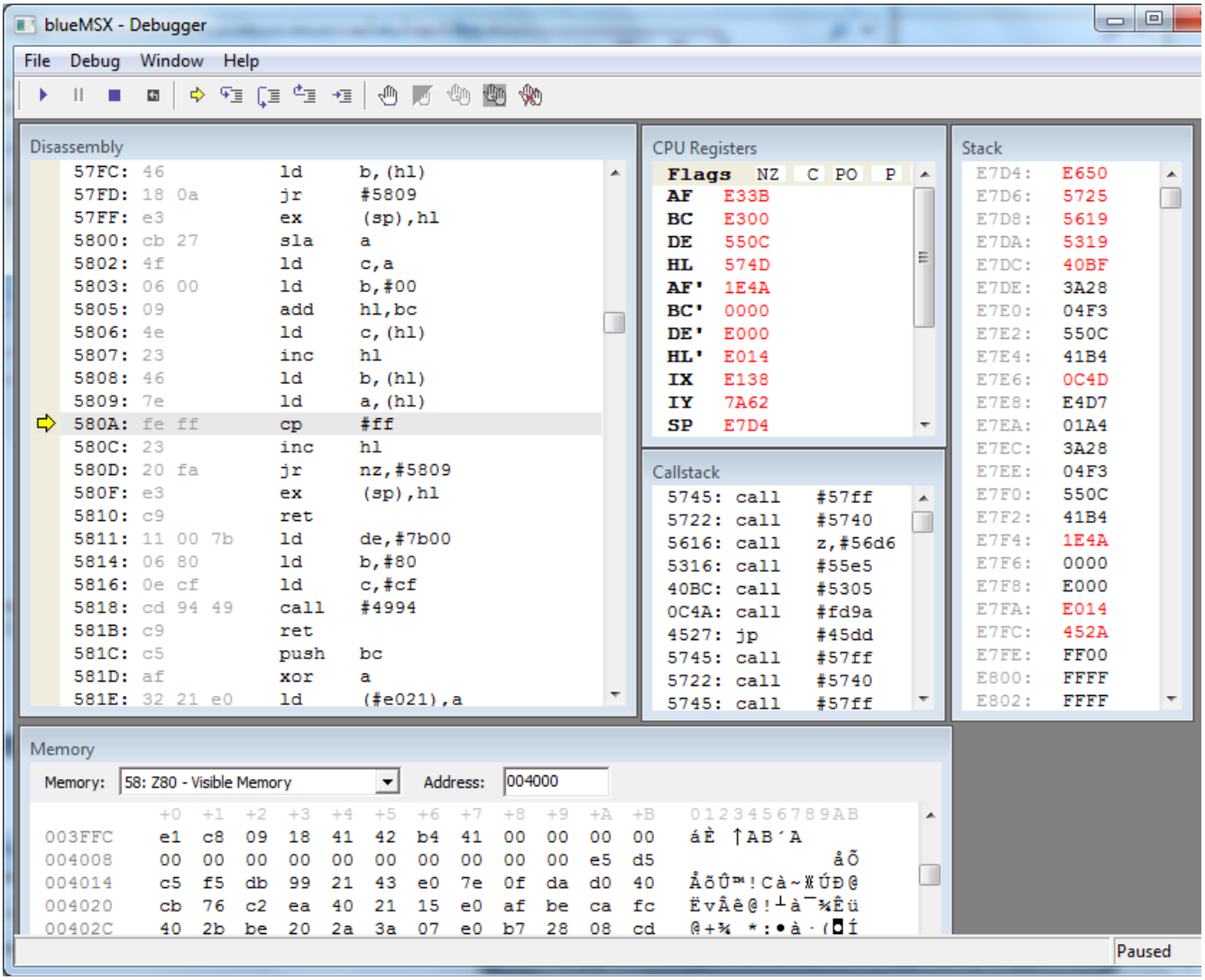


Este es el aspecto que tiene que tener el programa una vez que hemos abierto la ROM del Frogger de Konami. Aquí en el recuadro en Rojo esta el KIT de la cuestión todas las ROMS empiezan por los valores 41 42 seguido de la dirección donde empieza el código del programa de esta manera sabemos que el código 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).

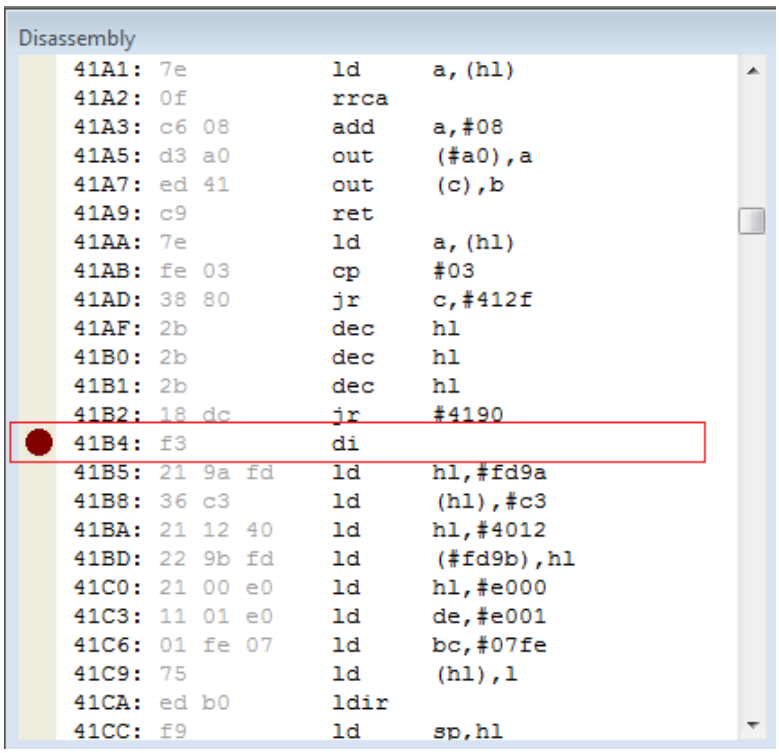
Ya podéis cerrar este programa ya que hemos averiguado nuestro primer objetivo saber el inicio del código. Ahora vamos a empezar a tracear el juego. Abre el blueMSX y carga la ROM del Frogger, después abre el debugger y dale al botón del Play  en el blueMSX para que se ejecute la ROM, veras que esta 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.



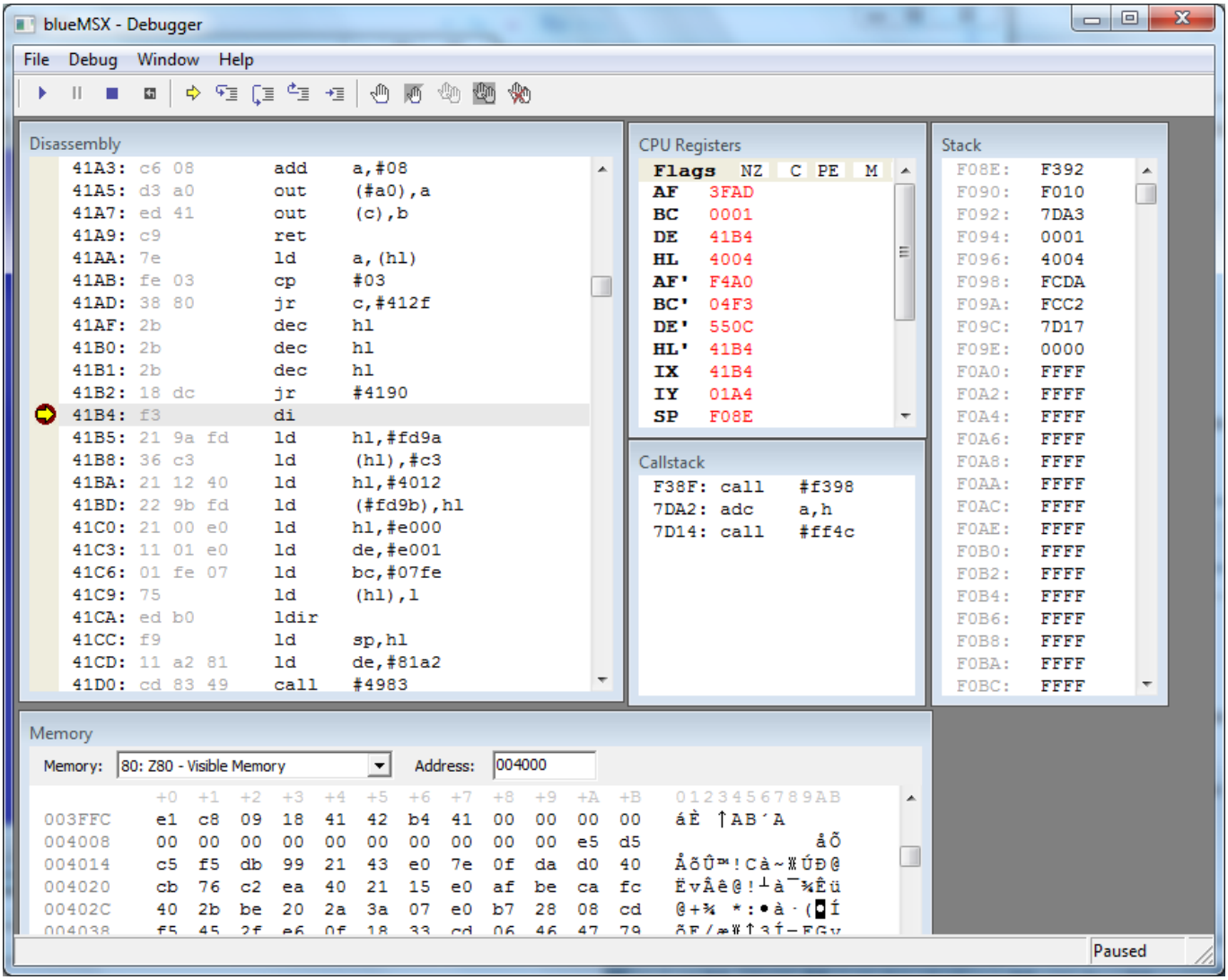
Ya tenemos el código detenido, ahora nos vamos a la posición de memoria 41B4H para colocar un punto de interrupción en esa dirección que es donde empieza el código juego Frogger.



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 hara 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)