asMSX

Ensamblador cruzado para MSX

v.0.12g

– Edición 2007 –

[18 de Marzo de 2007]

Indice

1. Introducción

1.1. Descripción

asMSX es un ensamblador para el procesador Z80 que además incorpora una serie de elementos que facilitan mucho la tarea de programar para MSX. Se trata de un ensamblador cruzado que se ejecuta en ordenadores PC con sistema operativo Windows. Gracias a la velocidad de los actuales ordenadores personales, el ensamblado es prácticamente instantáneo, lo cual supone una ventaja considerable respecto a los ensambladores nativos para MSX. Del mismo modo, no existe ninguna restricción para el tamaño del código fuente y se puede editar con cualquier editor o tratamiento de textos.

La primera versión de asMSX (v.0.01) fue publicada en el año 2000, y se trataba de un boceto de lo que es hoy asMSX. Los fallos de aquella versión original han sido subsanados, a la vez que se han ampliado ciertos aspectos y se le ha dotado de más características avanzadas.

1.2. Características

Entre los puntos a destacar de asMSX pueden citarse los siguientes:

- Soporta todas las instrucciones oficiales del Z80 con la sintaxis de Zilog.
- Ensambla todas las instrucciones no oficiales documentadas por Sean Young.
- Incluye además la sintaxis estandarizada del Z80 (acumulador implícito).
- Compatible con distintos sistemas de numeración (decimal, hexa, octal y binario).
- Operaciones aritméticas y lógicas incluidas a nivel de código fuente.
- Soporta números decimales con punto flotante, convertidos a punto fijo.
- Funciones matemáticas reales: trigonométricas, potenciales, etc.
- Acepta múltiples ficheros a través de inclusiones, permitiendo su anidado.
- Inclusión total o parcial de ficheros binarios externos inyectados directamente.
- Etiquetas locales y generales para el código.
- Rutinas de la BIOS del MSX predefinidas con sus nombres oficiales.
- Generación automática de ficheros binarios cargables desde BASIC.
- Producción automática de archivos ROM.
- Soporta 4 tipos distintos de megaROM: Konami, Konami SCC, ASCII 8 y 16 KB.
- Uso de variables numéricas internas del ensamblador.
- Ensamblado de ficheros COM para MSX-DOS.
- Exportación de tabla de símbolos (SYM).
- Impresión de textos indicativos desde el código (TXT).
- Integración con el debugger de BlueMSX.
- Ensamblado condicional mejorado
- Generación de ficheros CAS y ficheros WAV cargables en el MSX

1.3. Justificación

¿Es de verdad necesario otro ensamblador para Z80? Probablemente no. En los apéndices se incluye una lista de ensambladores para MSX, tanto cruzados como nativos, así como su disponibilidad. Entre ellos, hay herramientas de desarrollo excelentes que han sido utilizadas por muchos programadores.

El propósito de asMSX es proporcionar un ensamblador con soporte completo de las instrucciones del Z80, flexible y cómodo, fiable y orientado en gran medida al desarrollo de software para MSX. Estos objetivos se han alcanzado con la versión actual. Se trata del ensamblador más cómodo para el desarrollo específico de ROMs y MegaROMs para MSX.

El resultado es una aplicación para consola de Windows de unas 32 KB de tamaño y con una velocidad de ensamblado muy elevada.

1.4. Sintaxis

La implementación del lenguaje ensamblador para el microprocesador Z80 que hace asMSX diverge en algunos aspectos de la sintaxis oficial de Zilog. La mayor diferencia es que las indirecciones se hacen utilizando los corchetes, [ ], no los paréntesis ( ). El porqué de este cambio quedará claro al ver el potente lenguaje matemático que incorpora asMSX, que permite cálculos muy complejos, por lo que los paréntesis se emplean en la evaluación numérica.

1.5. Utilización

Se trata de una aplicación de consola para Windows, por lo que la forma más cómoda de utilizarla es desde una ventana del intérprete de comandos o MS-DOS. Para ensamblar un fichero basta con hacer lo siguiente:

```
asMSX fichero.ext
```

Si la extensión no se especifica, se tomará por defecto .asm. Los ficheros de código fuente deben ser ficheros de texto sin formato, codificado como ASCII de 8 bits. También es posible ensamblar el código fuente arrastrando el icono que contiene el programa hasta el icono del propio asMSX, aunque no se recomienda. Los nombres largos de Windows pueden no funcionar adecuadamente, por lo que se sugiere que no se incluyan puntos dentro del nombre del fichero, salvo el que indica el inicio de su extensión.

Durante el ensamblado aparecerán una serie de mensajes, y si no hay ningún error en el código, se generarán una serie de archivos:

- `fichero.sym`: un fichero de texto en modo ASCII que contiene todos los símbolos definidos en el programa así como su valor expresado en decimal. Este archivo es compatible con el emulador BlueMSX, cuyo debugger permite cargar tablas de símbolos externos para facilitar el depurado de programas.
- `fichero.txt`: otro fichero de texto en modo ASCII con las impresiones que se hayan hecho desde el programa. Si no se emplea ninguna de las instrucciones de tipo PRINT, no se generará.

- `fichero[.z80/.bin/.com/.rom]`: el resultado de ensamblar el código fuente se guardará con el mismo nombre que el archivo de entrada, eligiendo la extensión adecuada según las directivas que se empleen. Así, la extensión Z80 es para código sin cabecera, BIN es para los archivos binarios cargables desde MSX-BASIC con BLOAD, COM para los ejecutables de MSX-DOS, y ROM para las ROMs y megaROMs de MSX.

2. Lenguaje ensamblador

2.1. Sintaxis de lenguaje

El formato de línea en el código fuente que acepta asMSX es el siguiente:

```
[etiqueta:] [directiva[parametros]] [;comentario]
[etiqueta:] [opcode[parametros]] [;comentario]
```

Cada uno de estos bloques son opcionales, y se explican con mayor detalle en los siguientes apartados. No hay ningún tipo de limitación respecto a longitud de línea, espacios intermedios o tabulación, ya que el preprocesador de código se encarga de ajustar estos aspectos.

Ejemplos:

```
bucle:
ld
a,[0ffffh]
; lee el registro maestro
puntos:
db
20,30,40,50
; tabla de puntos
Org
08000h; posiciona el código en la página 2
Nop
```

2.2. Etiquetas

Las etiquetas son una forma de identificar una cierta posición de memoria. En realidad, equivale a dar un nombre a un valor de 16 bits. Los nombres para las etiquetas pueden ser alfanuméricos, aunque el primer carácter tiene que ser una letra o el carácter barra baja “_”. Para que quede definida como etiqueta, debe quedar seguida por dos puntos. Eso sí, las mayúsculas y las minúsculas se distinguen, es decir, no es lo mismo etiqueta: que ETIQUETA: o eTiQuEta:

Ejemplos:

```
ETIQUETA:
```

```
Etiqueta:
_alfanumerica:
bucle001:
```

Los dos puntos sirven para definir la etiqueta, cuyo valor equivaldrá a la posición de memoria a ocupar por la próxima instrucción. Para hacer uso del valor de la etiqueta, se debe referenciar sin incluir los dos puntos. Además, es posible definir etiquetas locales, precediendo las etiquetas con @@. La particularidad de las etiquetas locales es que sólo están disponibles entre una etiqueta global y otra. Por ello, es perfectamente lícito un código así:

```
RUTINA1:
...
@@BUCLE:
...
RUTINA2:
...
@@BUCLE:
...
```

2.3. Expresiones numéricas

Las expresiones numéricas pueden ser números o bien el resultado de operaciones más o menos complejas hechas con esos números.

2.3.1. Formatos numéricos

En cuanto a los sistemas de numeración compatibles, se indican a continuación junto con la sintaxis adecuada y algunos ejemplos:

- **DECIMAL**: los números en base 10 se expresan normalmente, como un grupo de uno o más dígitos decimales. La única restricción es que no se deben incluir ceros a la izquierda, es decir, se emplean tal y cómo se expresan habitualmente por escrito.

  Ejemplos:

  ```
  0    10    25    1    255    2048
  ```

- **DECIMAL NO ENTERO**: se expresan como los decimales enteros, pero separando con un punto la parte entera de la parte decimal. Es necesario que siempre esté presente ese punto como separador, porque en caso contrario se interpretará como un número entero.

  Ejemplos:

```
3.14    1.25    0.0    32768.0    -0.50    15.2
```

- **OCTAL**: los números en base 8 se pueden expresar de dos formas distintas. La primera es el formato que se utiliza en los lenguajes C, C++ o Java para indicarlos, es decir, precedidos por un 0 siempre y a continuación todos los dígitos octales (es decir, del 0 al 7). La otra forma es la habitual de los lenguajes ensambladores, es decir, el grupo de dígitos octales y seguido por una letra O, sea minúscula o mayúscula. Se han incluido por compatibilidad, pero no se trata de un sistema de numeración cómodo para una arquitectura como la del Z80.

  Ejemplos:

  ```
  01    077    010    1o    77o    10o
  ```

- **HEXADECIMAL**: los números en base 16, los más habituales al programar en ensamblador, pueden expresarse de tres formas diferentes. La primera de ellas es, de nuevo, compatible con la utilizada en los lenguajes C, C++ o Java, es decir, siempre precedidos por 0x y a continuación el grupo de dígitos hexadecimales (de 0 a 9 y de A a F). La segunda forma de expresarlos es la utilizada en el lenguaje Pascal: el grupo de dígitos hexadecimales va precedido por el símbolo $. Finalmente, se ha incluido también el formato más habitual de los números hexadecimales en ensamblador: el grupo de dígitos hexadecimales seguido por la letra H, sea minúscula o mayúscula. Nótese que en este caso, el primer dígito debe ser siempre numérico, es decir, que si el número hexadecimal empieza por una letra, habrá que añadirle un cero adicional a la izquierda.

  Ejemplos:

  ```
  0x8a    0xff    0x10    $8a    $ff    $10    8ah    0ffh    10h
  ```

- **BINARIO**: los números en base 2 tienen una única forma de expresión compatible con asMSX. Tras el grupo de dígitos binarios (cero y uno) debe ir la letra B, sea en minúscula o en mayúscula.

  Ejemplos:

  ```
  1000000b    11110000b    0010101b    1001b
  ```

2.3.2. Operadores

Además de los números, expresados en cualquiera de las formas vistas, se pueden realizar operaciones con ellos. La notación de los operadores es la habitual, y para las operaciones menos comunes se ha optado por seguir la sintaxis del lenguaje C/C++.

+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo (resto de división entera)

Adicionalmente, hay otras operaciones:

<< Desplazamiento a la izquierda (shift left). Desplaza a la izquierda el número de bits indicado, lo que equivale a multiplicar por dos el mismo número de veces.

>> Desplazamiento a la derecha (shift right). Desplaza a la derecha el número de bits indicado, equivale a dividir entre dos el número de veces dado.

Además de las operaciones aritméticas disponemos de las operaciones lógicas, que operan a nivel de bits. De nuevo, la sintaxis preferida es la del lenguaje C. Las operaciones binarias son las siguientes:

| O a nivel de bits(*OR*).

& Y a nivel de bits (*AND*).

^ O exclusivo a nivel de bits (*XOR*).

Y existe una operación unaria:

~ NO lógico (*NOT*). Realiza el complemento a uno.

También se han definido operaciones de comparación lógica, que utilizan también la misma sintaxis del lenguaje C:

|| O lógico (OR)

&& AND lógico

== Igualdad

!= Desigualdad

< Menor

<= Menor o igual

> Mayor

>= Mayor o igual

El orden de preferencia de los operadores es el habitual, el mismo empleado en C/C++. Además, se pueden utilizar los paréntesis para agrupar las operaciones, como se hace habitualmente con las operaciones ariméticas.

Ejemplo:

```
((2*8)/(1+3))<<2
```

En cuanto a los decimales no enteros, se pueden emplear las mismas operaciones indicadas antes y, además, otras funciones reales:

`SIN(X)`	Función del seno. Las unidades de entrada son los radianes.
`COS(X)`	Función del coseno.
`TAN(X)`	Función tangente.
`ACOS(X)`	Arcocoseno.
`ASIN(X)`	Arcoseno.
`ATAN(X)`	Arcotangente.
`SQR(X)`	Elevar al cuadrado.
`SQRT(X)`	Raíz cuadrada.
`EXP(X)`	Función potencial, e elevado a X.
`POW(X,Y)`	Elevar la base X al exponente Y.
`LOG(X)`	Logaritmo decimal.
`LN(X)`	Logaritmo neperiano.
`ABS(X)`	Devuelve el valor absoluto de X.

Adicionalmente, se ha definido por defecto `PI` con precisión doble, para poderlo utilizar directamente con las funciones reales el valor π.

Ejemplo:

```
sin(pi*45.0/180.0)
```

Para utilizar estos valores decimales dentro del programa en ensamblador para Z80 es indispensable convertir los números en punto flotante a punto fijo. Para ello, están disponibles las siguientes funciones de conversión:

`FIX(X)`	Convierte la punto flotante a punto fijo.
`FIXMUL(X,Y)`	Calcula el producto de dos números en punto fijo.
`FIXDIV(X,Y)`	Calcula la división de X entre Y.
`INT(X)`	Convierte a entero un número no entero.

Existe un símbolo que tiene siempre un valor cambiante, y es $. El valor del símbolo $ equivaldrá en todo momento a la dirección de ensamblado actual, o, lo que es lo mismo en términos de ejecución, al valor del registro PC.

2.4. Introducción de datos

Los datos se pueden incluir a través de distintas directivas:

```
db      dato,[dato...]
defb    dato,[dato...]
```

```
dt        "texto"
deft      "texto"
```

Con estas instrucciones se incluyen los datos como bytes, datos de 8 bits. DT se ha incluido por compatibilidad con otros ensambladores, pero en realidad, las cuatro instrucciones son equivalentes.

```
dw        dato,[dato...]
defw      dato,[dato...]
```

Así se incluyen datos como words, es decir, datos de 16 bits.

```
ds        X
defs      X
```

De este modo se reserva un espacio de X bytes en el punto actual de memoria. Sirve para identificar variables en RAM, por ejemplo. Para facilitar, los tamaños más habituales, es decir, byte y word, han sido predefinidos, pudiéndose usar las siguientes directivas:

`.byte`	reserva un espacio de un byte (8 bits)
`.word`	reserva un espacio de dos bytes (16 bits), es decir, un word.

2.5. Directivas

Las instrucciones predefinidas sirven para organizar el código y habilitar las características adicionales de asMSX.

- `.ORG X` Sirve para indicar una posición en memoria al ensamblador. El código subsiguiente se ensamblará a partir de esa dirección.
- `.PAGE X` Equivale a la instrucción anterior, pero no se indica una dirección sino un número de página de la memoria. Así, .PAGE 0 equivale a .ORG 0000h, y .PAGE 1 equivale a .ORG 4000h, .PAGE 2 equivale a .ORG 8000h y .PAGE 3 equivale a .ORG 0C000h.
- `Identificador .EQU expresión` Sirve para definir una constante. Las reglas de construcción del identificador son las mismas que para construir un nombre de etiqueta, si bien no es posible definer identificadores locales.
- `Variable = expresión` asMSX permite ahora el uso de variables enteras. Las variables deben ser inicializadas, y después se les puede asignar

valores numéricos y hacer operaciones con ellas. Por ejemplo, se podría realizar lo siguiente:

```
Valor1=0
Valor1=Valor1+1
Ld   a,Valor1
Valor1=Valor1*2
Ld   b,valor1
```

- `.BIOS` Predefine las direcciones de las rutinas de la BIOS, incluyendo las especificadas en los estándares MSX, MSX2, MSX2+ y Turbo-R. Se emplean los nombres habituales en mayúsculas.
- `.ROM` Define la cabecera para producir una ROM. Es imprescindible indicar antes la posición, que puede hacerse con la directiva .PAGE. También es conveniente indicar con la directiva .START el punto de inicio del programa.
- `.MEGAROM [mapeador]` Define la cabecera y estructura para producir una megaROM. Por defecto se define también la subpágina 0 del mapeador, por lo que no es necesario incluir ninguna instrucción ORG ni PAGE o SUBPAGE previa. Los tipos de mapeador soportado son los siguientes:
  - `Konami`: tamaño de subpágina de 8 KB, límite de 32 páginas. El tamaño máximo de la megaROM será de 256 KB (2 megabits). Entre 4000h y 5FFFh está necesariamente la subpágina 0, es decir, no puede cambiarse.
  - `KonamiSCC`: tamaño de subpágina de 8 KB, límite de 64 páginas. El tamaño máximo de la megaROM será de 512 KB (4 megabits). Soporta acceso a SCC, el Sound Custom Chip de Konami.
  - `ASCII8`: tamaño de subpágina de 8 KB, límite de 256 páginas. El tamaño máximo de la megaROM será de 2048 KB (16 megabits, 2 megabytes).
  - `ASCII16`: tamaño de subpágina de 16 KB, límite de 256 páginas. El tamaño máximo de la megaROM será de 4096 KB (32 megabits, 4 megabytes).
- `.BASIC` Genera la cabecera para producir un archivo binario cargable desde MSX-BASIC. Es necesario indicar antes una posición inicial con la directiva .ORG, y en el caso de que el principio de ejecución del programa no coincida con el principio del programa, utilizar también la directiva .START. La extensión por defecto del fichero de salida es BIN.
- `.MSXDOS` Produce como resultado un archivo COM ejecutable desde MSX-DOS. No es necesario utilizar la instrucción .ORG porque el inicio se establece en 0100h.
- `.START X` Indica la dirección de inicio de ejecución para archivos ROM, megaROM y BIN de no coincidir ésta con el principio del fichero.

- `.SEARCH` Para ROMs y megaROMs que arrancan en la página 1 (4000h), se encarga de buscar automáticamente el slot y subslot de la página 2 correspondiente (8000h). Equivale al siguiente bloque de código:

```
        call    0138h ;RSLREG
        rrca
        rrca
        and     03h
; Secondary Slot
        ld      c,a
        ld      hl,0FCC1h
        add     a,l
        ld      l,a
        ld      a,[hl]
        and     80h
        or      c
        ld      c,a
        inc     l
        inc     l
        inc     l
        inc     l
        ld      a,[hl]
; Define slot ID
        and     0ch
        or      c
        ld      h,80h
; Enable
        call    0024h ;ENASLT
```

- `.SUBPAGE n AT X` Esta macro se usa para definer las distintas subpáginas de una megaROM. En el modelo de generación de megaROMs de asMSX, todo el código y los datos deben incluirse en subpáginas, que equivalen a los bloques lógicos de operación del mapeador. Se debe indicar el número de subpágina que se desea crear, y en qué dirección lógica ensamblarla, es decir, en que posición se dará su ejecución.
- `.SELECT n AT x / .SELECT registro AT x` Como replica a la anterior, esta macro selecciona la subpágina n en la dirección x. El código concreto utilizado para esta operación dependerá del tipo de mapeador que se haya seleccionado. No altera el valor de ningún registro ni afecta al modo de interrupción ni a las banderas de estado.
- `.PHASE X / .DEPHASE` Estas dos rutinas permiten utilizan direcciones virtuales de memoria. Es decir, ensamblar en una posición de memoria instrucciones que luego se ubicarán en otra. Puede ser útil para introducir código en una ROM que luego se tenga que copiar y ejecutar desde la RAM. Su efecto es que las etiquetas se ensamblarán de acuerdo con la dirección dada. .DEPHASE revierte al estado normal, si bien cualquier instrucción ORG, PAGE, SUBPAGE tendrán el mismo efecto.
- `.CALLBIOS X` Llama a una rutina de la BIOS desde MSX-DOS. Equivale al siguiente bloque de código:

```
LD   IY,[EXPTBL-1]
LD   IX,RUTINA
CALL CALSLT
```

- `.CALLDOS X` Ejecuta una de las funciones del MSX-DOS. Equivale al siguiente código:

```
LD C,CODIGO
CALL 0005h
```

- `.SIZE X` Establece el tamaño del archivo resultante en Kilobytes.
- `.INCLUDE "archivo"` Incluye código fuente almacenado en un fichero externo.
- `.INCBIN "archivo" [SKIP X] [SIZE Y]` Inyecta un fichero binario externo dentro del programa. Se pueden utilizar adicionalmente los modificadores SKIP y SIZE, que permiten indicar el número de bytes a descartar desde el principio del archivo y el número de bytes a guardar en total.
- `.RANDOM(n)` Genera un valor aleatorio entero entre 0 y n-1. Permite una aleatoriedad mayor que el uso directo del registro R del Z80.
- `.DEBUG "texto"` Establece un texto dentro del programa para que pueda ser visto al hacer debug, que es transparente en cuanto a la ejecución. El debugger del emulador BlueMSX soporta funcionalidades adicionales.
- `.BREAK [X] / .BREAKPOINT [X]` Definen un breakpoint para el debugger del emulador BlueMSX. Su ejecución es trasparente, si bien se recomienda su eliminación en el fichero final. Si no se indica la dirección, se ejecutará el breakpoint en la posición en la que se haya definido.
- `REPT n / ENDR` Esta macroinstrucción permite repetir un número dado de veces un bloque de código. Admite anidaciones para generar automáticamente tablas complejas. La única restricción es que el número de repeticiones debe indicarse como un número decimal directo, no como una expresión numérica. Como ejemplos:

```
REPT  16
  OUTI
ENDR

X=0
Y=0
REPT 10
   REPT 10
      DB X*Y
      X=X+1
   ENDR
```

```
        Y=Y+1
    ENDR
```

- `.PRINTTEXT "texto" / .PRINTSTRING "texto"` Imprime un texto en el fichero de texto de salida.
- `.PRINT expresión / .PRINTDEC expresión` Imprime un valor numérico en formato decimal.
- `.PRINTHEX expresión` Imprime un valor numérico en formato hexadecimal
- `.PRINTFIX expresión` Imprime un valor numérico en punto fijo.
- `.CAS [“texto”] / .CASSETTE [“texto”]` Genera un fichero CAS con el resultado del ensamblado. Sólo válido para formato BASIC, ROM en página 2 y Z80 directo. El texto permite indicar un nombre para la carga desde cassette, con un máximo de 6 caracteres de longitud. Si no se indica, se tomará el nombre del fichero de salida.
- `.WAV [“texto”]` Como el comando anterior, pero en lugar de generar un fichero CAS, genera un fichero WAV de audio, que puede ser cargado directamente en un MSX real a través del puerto de cassette.

2.6. Comentarios

Es muy recomendable utilizar comentarios a lo largo del código fuente en ensamblador. asMSX permite introducir comentarios en distintos formatos:

- `; Comentario`

  Comentario en una única línea. El punto y coma es el indicador estándar para comentarios en listados en ensamblador. La línea completa se tomará como un comentario, hasta el retorno de línea.

- `// Comentario`

  Funciona igual que el anterior. La doble barra es el indicador de comentario de una línea en C/C++.

- `/* Comentarios */`
- `{ Comentarios }`

  Comentarios en múltiples líneas. Todo el texto encerrado entre los delimitadores se interpreta como comentario y no será ensamblado.

2.7. Ensamblado condicional

asMSX incorpora soporte preliminar para ensamblado condicional. El formato de un bloque de ensamblado condicional es el siguiente:

```
IF   condición
     Instrucciones
ELSE
     Instrucciones
```

```
ENDIF
```

La condición puede ser de cualquier tipo, coherente con las reglas de formación de condiciones en C/C++ o Java. Es decir, una condición será cierta si su evaluación es distinta de cero.

Si la condición es cierta, se ensamblará el código siguiente al IF. Si la condición no es cierta, se omitirá su ensamblado y se ensamblará el código recogido a partir del ELSE.

Por supuesto, el bloque ELSE es opcional y no es necesario incluirlo en toda sentencia IF. Sin embargo, ENDIF tiene que ser utilizado para cerrar todo bloque condicional.

En esta nueva versión es posible anidar sentencias IF sin limitaciones, tal y como se muestra en el ejemplo siguiente:

```
IF (ordenador==MSX)
     call MSX1_Init
ELSE
     IF (ordenador==MSX2)
          call MSX2_Init
     ELSE
          IF (ordenador==MSX2plus)
               call MSX2plus_Init
          ELSE
               call TurboR_Init
          ENDIF
     ENDIF
ENDIF
```

Además, todo el código, directivas y macros serán ejecutadas según las condiciones, por lo que es posible crear estructuras del tipo siguiente:

```
CARTUCHO=1
BINARIO=2

formato=CARTUCHO

IF (formato==CARTUCHO)
     .page     2
     .ROM
```

```
ELSE
     .org 8800h
     .Basic
ENDIF

.START Init
```

La única limitación es que las sentencias IF <condición>, ELSE y ENDIF deben aparecer en líneas separadas, es decir, no pueden agruparse con otras instrucciones o macros en una misma línea de código. El siguiente código produciría un error al ensamblarse:

```
IF (variable) nop ELSE inc a ENDIF
```

Debería escribirse como sigue:

```
IF (variable)
     nop
ELSE
     inc  a
ENDIF
```

Adicionalmente, se ha incluido un nuevo tipo de condición, `IFDEF`, que no verifica el valor de la expresión utilizada, sino si dicha etiqueta se ha definido o no. Por ejemplo, el siguiente bloque:

```
IFDEF pruebas
     .WAV “Prueba”
ENDIF
```

Este fragmento de código hará que se genere un archivo WAV si y sólo si la etiqueta o símbolo `pruebas` ha sido definido previamente en el código fuente. Nótese que IFDEF sólo reconocerá una etiqueta como definida si se ha incluido antes de la sentencia IFDEF.

3. Herramientas adicionales

3.1. RLEpack

RLEpack es un sencillo compresor de datos que utiliza el sistema conocido como RLE, run-length encoded.

La compresión lineal es muy ineficiente y no se recomienda para datos con gran variabilidad, pero si hay grandes masas de datos homogéneos sí puede ser una solución a considerar. Aunque la compresión que consigue sea muy variable, la ventaja que tiene es que resulta muy sencillo y fácil de comprender.

Para obtener una compresión mucho más eficiente, se recomienda utilizar BitBuster, el compresor desarrollado por el TEAM BOMBA.

La idea de la compresión lineal es muy sencilla: si encuentra una serie de datos iguales, los cuenta e indica cuantos bytes seguidos tienen el mismo valor. El código fuente del algoritmo de descompresión se ha incluido en dos formatos distintos: uno para descomprimir el formato RLE a la memoria RAM y otro para descomprimir los datos a la VRAM directamente, sin pasar por la RAM principal.

Para comprimir un archivo binario, basta con pasar el nombre del fichero como parámetro único del ejecutable compresor, RLEpack. La aplicación generará un archivo con el mismo nombre y una extensión adicional .RLE (*run-length encode*). Adicionalmente, indicará el tamaño del archivo original y el tamaño del archivo comprimido, así como el ratio de compresión.

Las rutinas de descompresión son muy sencillas y se listan a continuación:

```
DEPACK_RAM:
; Descomprime datos comprimidos con RLEpack de RAM a RAM
; Parámetros: HL=dirección de los datos comprimidos
;DE=dirección donde situar los datos
@@LOOP:
ld a,[hl]
inc hl
cp 0C0h
jr c,@@SINGLE
and 3Fh
inc a
ld b,a
ld a,[hl]
inc hl
@@DUMP:
ld [de],a
inc de
djnz @@DUMP
jr @@LOOP
@@SINGLE:
ld c,a
or [hl]
ld a,c
ret z
ld b,1
```

```
jr @@DUMP

DEPACK_VRAM:
; Descomprime datos comprimidos con RLEpack de RAM a VRAM
; Parámetros: HL=dirección de los datos comprimidos
;
DE=dirección donde situar los datos en VRAM
; Nota: Conviene ejecutarla justo después del retrazado
ex de,hl
call SETWRT
ex de,hl
@@LOOP:
ld a,[hl]
inc hl
cp 0C0h
jr c,@@SINGLE
and 3Fh
inc a
ld b,a
ld a,[hl]
inc hl
@@DUMP:
out [98h],a
djnz @@DUMP
jr @@LOOP
@@SINGLE:
ld c,a
or [hl]
ld a,c
ret z
ld b,1
jr @@DUMP
```

Los ejemplos de código incluidos junto con esta distribución (ver apéndices) utilizan estas rutinas para gráficos que previamente fueron convertidos con PCX2MSX, comprimidos con RLEpack y añadidos al listado de código fuente mediante binDB.

Para comprimir un fichero basta con pasar el nombre del fichero como parámetro al programa RLEpack, del modo siguiente:

```
RLEpack fichero.ext
```

Se generará entonces el fichero comprimido fichero.ext.rle.

3.2. binDB

BinDB es una pequeña aplicación que convierte ficheros binarios a un listado en ensamblador en la forma de datos en forma de bytes, introducidos con la directriz DB. Aunque la directiva INCBIN permite incluir ficheros binarios dentro del código fuente de un programa en ensamblador, pero esto obliga a introducirlos íntegramente y no se

pueden modificar fácilmente, además de depender el ensamblado de varios ficheros. Para convertir un fichero binario a cadena de datos basta con pasar el nombre del fichero como parámetro al programa binDB, del modo siguiente:

```
binDB fichero.ext
```

Esto dará como resultado un nuevo archivo, fichero.ext.asm, conteniendo el binario como un listado hexadecimal en formato DB, con ocho bytes por línea, muy parecido a lo siguiente:

```
; fichero.ext - binary dump
; generated by binDB v.0.10
db 000h,000h,000h,000h,000h,000h,000h,000h
db 000h,000h,000h,000h,000h,000h,000h,000h
...
```

3.3. PCX2MSX

Este programa realiza la conversión de un fichero gráfico en formato PCX a dos ficheros separados, uno con la información de patrones y otro con la información de colores, necesarios para utilizar esos gráficos en el modo gráfico del alta definición del procesador de vídeo TMS9918/9928 y compatibles.

Como con las herramientas anteriores, el programa se emplea pasando el nombre del archivo a convertir como parámetro único en el comando de línea. Es decir, para convertir el archivo EJEMPLO.PCX, debe ejecutarse el siguiente comando:

PCX2MSX EJEMPLO.PCX

Y esto daría como resultado dos nuevos archivos, EJEMPLO.PCX.CHR y EJEMPLO.PCX.CLR, conteniendo los datos de los patrones y los datos de color, respectivamente.

La aplicación no realiza el ajuste gráfico de atributos, así que sólo convertirá archivos que cumplan con la limitación de colores de SCREEN 2, es decir, dos colores por cada bloque de ocho píxeles horizontales. En caso contrario, si existe algún problema de atributos, el programa lo detectará e indicará la posición concreta en la que ha localizado el error. Esto daría el error siguiente:

```
Color collision at line (X,Y)
```

Nótese que la conversión no tendrá en cuenta la paleta de color definida en el archivo PCX, sino simplemente el orden. Es decir, la conversión final utilizará el orden de color de la paleta del procesador TMS9918/9918.

En cuanto al archivo PCX, debe estar en formato de 8 bits, es decir, 256 colores y con compresión RLE, lo que es lo más habitual. Tanto da que tenga información de color o no, la conversión se hará en base al orden de colores.

Nota: algunos programas de edición gráfica modernos soportan el formato PCX pero de una forma un tanto heterodoxa, invirtiendo el orden normal de los colores. Por ello, además de la versión normal del programa, PCX2MSX, hay una versión alternativa, PCX2MSXi, que permite la conversión de archivos PCX con paleta invertida.

Se incluye un fichero PCX de ejemplo con los 16 colores del procesador de video TMS9918/9928 predefinidos (calibración aproximada) y con la paleta invertida, llamado SAMPLE.PCX.

4. Código fuente

4.1. Classic Pong v.0.15

Aquí se presenta una versión para MSX de uno de los juegos electrónicos más antiguos: el PONG, cuya versión arcade fue producida inicialmente por Atari Games. Se trata de una versión muy simplificada, exclusivamente para dos jugadores, pero que supone un ejemplo lo suficientemente amplio como para poder explorar las capacidades del asMSX y ver cómo se estructura un juego completo en ensamblador para MSX.

Las instrucciones de juego son muy sencillas: el primer jugador que consiga marcar diez goles gana la partida. El saque es aleatorio y automático, y cada jugador controla su pala: la de la izquierda, del *PLAYER ONE* o jugador uno, con las teclas del cursor o el joystick en el puerto 1. La pala de la derecha, del *PLAYER TWO* o jugador dos, exclusivamente con el joystick en el puerto 2.

Las técnicas que se utilizan en este programa son muy sencillas y el código está repleto de comentarios e indicaciones al respecto. La idea de publicar juegos completos con su código fuente responde a la intención de que los programadores noveles puedan acceder de forma fácil al mundo del ensamblador para MSX y puedan alterar el código a su gusto y ver rápidamente los resultados. Como programador original de este juego sí que creo que hay varias mejoras evidentes que hacer, que se dejan como ejercicio a los programadores interesados: completar el sonido, utilizar gráficos coloristas (esta versión pretende ser respetuosa con el original, en blanco y negro), la opción de jugar contra el ordenador y que éste tenga distintos niveles de inteligencia, etc.

La única condición de utilización del código fuente es que cualquier versión modificada del juego se distribuya también libremente y con el código fuente e incluya también el código fuente de la versión original. Las versiones que tengan el nivel de calidad suficiente y supongan una mejora sobre ésta, podrían ser incluidas en futuras distribuciones de asMSX o en la página oficial de distribución del asMSX, siempre y cuando el autor de dichas modificaciones o extensiones esté de acuerdo con ello.

4.2. Classic Minesweeper v.0.10

Otro juego clásico, que nunca fue versionado para MSX de la primera generación, pasa por fin a engrosar el listado de programas disponibles para los ordenadores de la norma. Además, de nuevo, lo hace en forma opensource, es decir, se trata de un programa con código abierto, listo para ser ampliado, modificado o alterado libremente por cualquier usuario, siempre y cuando respete las condiciones de uso: el resultado debe seguir siendo de libre distribución y el código nuevo debe ser acompañado por el código de la versión original. Este juego no requiere de ningún tipo de presentación,

porque probablemente sea uno de los juegos más instalados del mundo, ya que ha acompañado a todas las versiones del sistema operativo Windows, además de haber sido versionado para el resto de sistemas operativos compatibles con ordenadores PC.

Para hacerlo compatible con las configuraciones más habituales de MSX, esta versión del Buscaminas se controla mediante teclado o joystick, no siendo compatible con el ratón. Los controles son sencillos: se emplean las teclas direccionales del cursor o la palanca del joystick conectada al puerto 1. El equivalente al click del botón izquierdo del ratón en la versión del PC es aquí la barra espaciadora o el botón 1 del joystick, y supone descubrir la ficha a la que apunta el cursor. En caso de ser una mina, el juego habrá terminado: *game over*.

La función del botón derecho del ratón se ha mapeado como la tecla control del teclado o el botón 2 del joystick, y su función es la de marcar la casilla actual con una bandera de señales, o bien, marcarla como interrogante o eliminar todas las marcas anteriores. El juego acaba con éxito cuando todas las minas del campo han sido correctamente balizadas con banderas de señales.

En la versión de PC se permite hacer un click simultáneo con los botones derecho e izquierdo del ratón sobre las celdas con número una vez que están todas las banderas necesarias a su alrededor marcadas. Se permite así un avance rápido, al ser el equivalente a descubrir una por una todas las celdas adyacentes. Esta función ha sido incluida en esta versión del Buscaminas del mismo modo, pulsando simultáneamente el boton 1 y 2 del joystick, o bien, las teclas control y barra espaciadora.

Desde el menú principal se puede ajustar el nivel de dificultad, eligiendo desde 50 hasta 175 minas, para un campo de juego de tamaño fijo, 30 casillas de ancho por 20 de alto. Por supuesto, el nivel indicado como más fácil es verdaderamente fácil, mientras que el que aparece señalado como imposible es poco menos que imposible, sí. Además de saber jugar y tener una mente ágil, un poco de suerte es siempre imprescindible en los niveles más avanzados del juego.

En cuanto al juego, es completo y funcional, tiene algunos detalles interesantes, tanto para el tratamiento gráfico como respecto a técnicas de programación en general, como procedimientos recursivos, uso de registros índice, generación de números pseudos aleatorios, etc. Sin embargo, puede entenderse como un nuevo ejercicio para programadores con iniciativa y ganas de aprender: el aspecto gráfico y sonoro son mejorables (faltan efectos especiales, no hay melodías, etc.), se pueden introducir nuevos modos de juego o un sistema más cómodo de menús y configuración... Las posibilidades son casi infinitas: dependen de lo que cada uno crea que puede aportar a un juego así. Para los detal es de la licencia de uso de este programa, véase la licencia de uso del asMSX y los comentarios incluidos en el apéndice referido al juego Classic Pong v.0.15.

4.3. Otros ejemplos

Hay otros ejemplos de código fuente bastante más sencillos, que se han incluido para detallar algunas de las capacidades del asMSX, o como demostraciones de alguna funcionalidad adicional.

- OPCODES.ASM: ejemplo de ensamblado que incluye todas las instrucciones documentadas e indocumentadas del procesador Z80, con las sintaxis oficiales de Zilog y las alternativas (acumulador implícito, sintaxis normalizada, etc.).

- DATA.ASM: ejemplo de ensamblado que cubre los diferentes formatos de datos que soporta asMSX, tanto numéricos como alfanuméricos.
- MEGAROM.ASM: ejemplo de uso de las nuevas directivas del asMSX para hacer una megaROM muy fácilmente.
- FIXED.ASM: sencilla demostración de uso de aritmética de punto fijo y las funciones matemáticas incluidas en asMSX, con movimientos complejos para sprites multicolores.
- TV.ASM: pequeña demostración en la que aparece la imagen de un televisor sin sintonizar, con la característica nieve sin patrones repetidos.

4.4. Más ejemplos

Si alguien ha desarrollado algún programa con asMSX que considere interesante y del que otras personas puedan aprender, queda invitado a mandarlo a la página de MSX Workshop mediante la siguiente dirección de correo electrónico: asMSX@robsy.net.

La otra opción es que lo haga llegar directamente a los foros de Karoshi Corporation, http://forum.karoshicorp.com , donde todos los desarrolladores son bienvenidos.

5. Información adicional sobre asMSX

5.1. Novedades en la versión 0.12g

- Corrección de un error asociado al uso de etiquetas locales tras utilizar la instrucción PHASE.
- Soporte preliminar para cassette con las directivas CAS y WAV – generación de archivos CAS [emuladores] y WAV [MSX real]
- Mejora del ensamblado condicional con IFDEF.

5.2. Novedades en la versión 0.12f

- Corrección de un error asociado al cambio de versión que afectaba a los operadores binarios ^ (XOR) y | (OR).
- Soporte preliminar para ensamblado condicional.

5.3. Novedades en la versión 0.12e

Tras perder el código fuente de la versión 0.11, se ha tenido que reprogramar gran parte del código. Como novedades específicas de esta versión, se pueden citar las siguientes:

- Soporte nativo para megaROMs: MEGAROM, SUBPAGE y SELECT.
- Repeticiones con la macro REPT n / ENDR.
- Parámetros SKIP y SIZE para la instrucción INCBIN.
- Adecuación al emulador BlueMSX: DEBUG y BREAKPOINT.
- Definición del símbolo $, equivalente al valor del PC.
- Uso de variables numéricas a nivel de código.
- Generación de números aleatorios con RANDOM(n)

- Localización automática de la segunda página de ROMs de 32 KB con SEARCH.

Al tratarse de una versión experimental, podría haber problemas de inestabilidad. Por favor, reporta cualquier fallo detectado a asmsx@robsy.net.

5.4. Novedades en la version 0.11

Pocos cambios respecto a la version 0.10, que era muy estable. Se trata de una versión de mantenimiento. La lista completa de cambios es la siguiente:

- Soporte para desplazamientos negativos en las instrucciones que emplean los registros indice de 16 bits, IX e IY.
- Normalización del formato del archivo de símbolos y de texto, pasando de la sintaxis de Borland a la estándar para ensambladores respecto a los números en formato hexadecimal.
- Nuevas herramientas de apoyo incluidas en la distribución del ensamblador.
- Código fuente completo de juegos para MSX, comentados en profundidad.
- Documentación corregida y ampliada.

5.5. Novedades en la versión 0.10

El código de la versión 0.10 ha sido reescrito, por lo que los cambios sobre la primera versión publicada, 0.01, son innumerables y no merece la pena incluir un listado exhaustivo. Como características propias, se ha incluido el soporte para número en punto fijo y el formato de salida ROM.

5.6. Licencia de uso

Este programa queda declarado como software de libre copia y distribución, siempre que ésta sea gratuita. Para su venta o inclusión en colecciones de programas que se comercialicen es indispensable obtener la autorización del autor. En cualquier caso, se debe distribuir el ejecutable junto con la documentación y ejemplos que lo acompañan.

asMSX se puede utilizar en todo tipo de proyectos, sean comerciales o de libre distribución, sin otra obligación que acreditar en la documentación que se ha utilizado este ensamblador y dónde se puede obtener la versión más actual. Las herramientas incluidas junto con asMSX quedan sujetas a los términos de uso de la licencia principal.

5.7. Descargo legal

El autor renuncia a toda responsabilidad derivada del uso o funcionamiento de este programa. La utilización de este programa no supone el establecimiento de ningún tipo de relación contractual entre el usuario final y el programador. No se aceptará ninguna responsabilidad al respecto.

MSX es una marca registrada de la *MSX Association* de Japón.

6. Recursos disponibles

6.1. Páginas web de interés

- **MSX Info Pages** : http://www.hansotten.com : página personal de Hans Otten, *HansO*, donde encontraréis la mayor colección de documentaciones técnicas

originales escaneadas, esquemas, código fuente en ensamblador y una buena muestra del resto de ensambladores existentes para MSX.

- **Sean Young's MSX pages** : http://www.msxnet.org : página de Sean Young, el hombre sin el cual este ensamblador sería incompleto. Sus documentaciones originales sobre el Z80 son las más completas disponibles. Además, encontraréis documentaciones sobre distintos chips del MSX y artículos sobre megaROMs, SCC y FM-PAC.
- **MSX Assembler Page (MEP)** : http://map.tni.nl: una página orientada exclusivamente al ensamblador para MSX. Encontraréis aquí gran cantidad de información técnica sobre hardware del MSX y artículos sobre técnicas de programación. Imprescindible.
- **FUNET** : ftp://nic.funet.fi [*login anonymous*] : aquí lo encontraréis todo (o casi). Multitud de documentaciones técnicas escaneadas, programas con código fuente, utilidades adicionales, etc.
- **Robsy's MSX Workshop** : http://www.robsy.net : la antigua página oficial de asMSX, donde os podréis descargar ejemplos de código y algunas documentaciones exclusivas.
- **MSX Banzai!** : http://banzai.tni.nl : la página web de Patriek Lesparre, más conocido como GuyveR800. Tiene información interesante sobre el MSX en general y un breve F.A.Q. con técnicas de programación muy interesantes.

6.2. Otros recursos en Internet

Además de las páginas web dedicadas en concreto al desarrollo para el sistema MSX, existen otros muchos recursos disponibles sobre el ordenador MSX y que pueden ser de interés o ayuda para los usuarios de asMSX:

- **MSX Resource Center**: http://www.msx.org : el punto de reunión de usuarios de MSX a nivel mundial. Además de ser una fuente de noticias inacabable y muy frecuentemente actualizada, cuenta con los foros de MSX más activos. Muchos buenos programadores consultan regularmente la página, asi que pueden resolver vuestras dudas sobre programación o cualquier otro tema relacionado con el MSX. Las páginas están disponibles en inglés, holandés, portugués y español. En cualquier caso, el foro en inglés es el más internacional y en el que hay más movimiento.
- **MSX Posse**: http://www.msxposse.com : otro foro alternativo sobre el mundo del MSX. Sin ser tan grande como MRC, reúne ya a un nutrido grupo de desarrolladores y colaboradores fijos. Además, sus políticas sobre participación en los foros son bastante más relajadas. Este foro está disponible en inglés y holandés.
- **Foros de Karoshi Corporation**: http://forum.karoshicorp.com : un foro centrado básicamente en el desarrollo de video-juegos, con discusiones técnicas muy interesantes y repositorio de código en ensamblador.
- **HispaMSX**: http://www.hispamsx.org : un foro para la comunidad hispana del MSX. Sirve de soporte para la lista de correo HispaMSX, el punto de encuentro de todos los usuarios de habla hispana. La lista de correo está hospedada en

YahooGroups, y puede llegarse hasta ella a través de la dirección http://hispamsx.yahoogroups.com

6.3. Bibliografía recomendada

En internet se puede encontrar todo lo necesario para aprender a programar el microprocesador Z80 y, por extensión, los ordenadores MSX. De todos modos, existen libros excelentes sobre todos estos aspectos, entre los que resultan imprescindibles los siguientes:

- **Z80 Assembler, Rodnay Zaks**. Es la biblia del Z80, con una muy buena referencia (ahora desfasada respecto a las investigaciones de Sean Young) y con técnicas de programación básicas y avanzadas debidamente comentadas. Existe una edición en castellano publicada por Anaya. Existen otros libros sobre el Z80, pero éste es con mucha diferencia el mejor.
- **MSX Red Book, Avalon Software**. Un compendio estupendo sobre los MSX de la primera generación, con gran detalle de información sobre el hardware del MSX y su BIOS. No trata sobre el Z80, cuya programación se supone ya sabida. Existe una versión digital del mismo, que podéis consultar aquí: http://www.robsy.net/tredbook.txt
- **MSX Technical Databook, ASCII**. Un libro difícil de encontrar y que, tras mucho trabajo de búsqueda y edición, ya está disponible en internet. Contiene la definición del estándar MSX (primera generación), tratando tanto del sistema básico como del sistema extendido, con unidad de disco, puerto de serie, etc. Se puede descargar desde: http://www.robsy.net/msxtech.pdf
- **MSX2 Technical Book, ASCII**. La biblia del MSX2. Este también sería un libro difícil de encontrar, de no ser por el trabajo de Néstor Soriano, *Konamiman*, que lo tecleó entero. Podéis descargarlo directamente desde su página: http://www.konamiman.com

Además, es muy recomendable conseguir las documentaciones técnicas sobre los distintos procesadores programables: VDP, PSG, PPI, FM-PAC, SCC, etc. Entre ellos se encuentran los siguientes:

- **Z80 undocumented documented**. El documento más completo sobre el funcionamiento interno del Z80, mejor incluso que la información original publicada por Zilog. Un trabajo maestro de ingeniería inversa hecho por Sean Young. Se puede descargar desde su página o desde http://www.robsy.net/z80undoc.pdf
- **Z80 family: CPU user manual**. Este es el manual original de Zilog, con información relevante sobre el microprocesador Z80 tanto a nivel electrónico como a nivel lógico. Muy interesantes los ejemplos y los comentarios a las distintas instrucciones. Aun así, en caso de duda, es más fiable el documento de Sean Young. Este documento está en el propio site de Zilog, http://www.zilog.com , o más fácilmente localizable en la página de descarga de documentos de Robsy's MSX Workshop: http://www.robsy.net/um0080.pdf
- **Texas Instruments TMS9918A/TMS9928A/TMS9929A Video Display Processors**. La información original de Texas Instruments para los procesadores de video utilizados en los ordenadores MSX de la primera generación. Cubre con gran detalle multitud de aspectos sobre el motor gráfico

común a los MSX y a otros muchos ordenadores y consolas. Disponible para su descarga en http://www.robsy.net/tms9918.pdf

6.3. Otros ensambladores

Aquí se ha incluido una lista no exhaustiva de otros ensambladores para Z80. La primera incluye ensambladores cruzados, y la otra, ensambladores nativos para MSX. Existen muchos más, aquí sólo se citan los más habituales.

- **TNIasm - The New Image - Patriek Lesparre**: un completo ensamblador cruzado para Z80/R800/GBZ80.
- **SjAsm - XL2s - Sjoerd**: otro ensamblador cruzado que se actualiza muy a menudo.
- **Chaos Assembler 3 - Teddywarez**: entorno de desarrollo completo basado en el ensamblador TASM (*Table Assembler*).
- **TASM - Table Assembler**: un ensamblador cruzado con capacidad para producir código para multitud de procesadores distintos. No debe confundirse con el *Turbo Assembler* de *Borland*, también llamado TASM.
- **A22I - nocash - Martin Korth**: el ensamblador cruzado más compacto que se pueda encontrar, sólo 5 KB. Muy, muy rápido.
- **DEVPAC - Hisoft**: un paquete de desarrollo muy completo que incluye un ensamblador (GEN), un monitor (MON) y un editor para la versión de disco. Existe una versión para disco, CP/M, y otra para cargar desde cinta.
- **M80/L80 - Microsoft**: un ensamblador muy habitual, compuesto por un ensamblador y un linkador.
- **ZEN/CHAMP**: más de lo mismo.
- **RSC/RSC-II** - Ramón Salas: un ensamblador muy popular en España, que fue distribuido por la desaparecida MSX Club y Manhattan Transfer. A diferencia de los anteriores, incorporaba su propio editor y no requería MSX-DOS o CP/M.
- **WBASS2 - Door Wilbert Berendsen** : un ensamblador con muy buena fama, y de los más actualizados para MSX.
- **COMPASS - Compjoetania**: según muchos, el mejor ensamblador para MSX. Es comercial y hay una nueva versión anunciada desde hacer años, que parece que no saldrá.
- **AS/LD - Egor Voznessenski**: una obra maestra en cuanto a programación. De los mejores ensambladores disponibles para MSX-DOS. Necesita una máquina rápida para que el proceso de ensamblado no se haga eterno.

6.4. Consejos gratuitos

Aprender a programar en ensamblador, pese a lo que pueda parecer, no es difícil. Lo único que se requiere es tiempo y paciencia a partes iguales. Los materiales para aprender están disponibles en Internet, y los programas necesarios, también. Lo único de verdad imprescindible es vuestro entusiasmo.

7. Distribución

La distribución actual de asMSX v.0.12g debe incluir los siguientes archivos:

- *Ensamblador y documentación*
  - `asMSX.exe` - ensamblador cruzado.
  - `asMSX.pdf` - manual actualizado en castellano
- *Programas de ejemplo: directorio EXAMPLES*
  - `opcodes.asm` - todas las instrucciones del Z80, prueba de ensamblado.
  - `data.asm` - ejemplo de introducción de constantes, prueba de ensamblado.
  - `fixed.asm` - ejemplo básico de utilización de aritmética de punto fijo.
  - `tv.asm` - efecto de televisión sin sintonizar.
  - `unpack.asm` - rutinas de descompresión del formato RLE.
  - `pong.asm` - sencillo juego completo en formato ROM, con código comentado.
  - `mine.asm` - una versión para MSX del clásico "Buscaminas" en formato ROM.
  - `megarom.asm` – un ejemplo de creación de una megaROM sencilla. Importante: para cargar esta ROM en BlueMSX es imprescindible fijar manualmente el tipo de mapeador: ASCII 8 KB.
- *Herramientas adicionales: directorio TOOLS*
  - `RLEpack.exe` - compresor de ficheros utilizando un algoritmo lineal.
  - `binDB.exe` - convierte ficheros binarios a listado ensamblador, empleando DBs.
  - `PCX2MSX` - convierte ficheros gráficos en formato PCX a SCREEN 2.

Todas estas herramientas llevan su propia documentación adjunta, incluida en los apéndices de esta documentación.