Manual de uso del compilador SjasmPG

Esta es una versión del compilador sjasm 0.42c de XL2S Entertainment modificada y mejorada por PipaGerado.

Línea de comandos:
sjasmpg[win|linux|mac]_[esp|eng][.exe] [-opciones] archivo_entrada [archivo_salida]
Opciones:

-s	Genera un archivo sym para el debug del blueMSX
-q	No genera listados informativos de salida
-i<path>	Ruta de Inclusión
-j	Optimiza los saltos (jump)(JP, JR, etc)
-p	Usa los paréntesis como indirecciones (no lo recomiendo).

Ejemplo:
sjasmpg_win_esp.exe -s main.asm main.rom

Sintaxis: Se puede usar las operaciones del Z80 oficiales y no oficiales en minúsculas o en mayúsculas (no mezcladas), pero **nunca deben ir al principio de una línea** porque esto se reserva para las etiquetas las variables y las directivas del preprocesador. Se usan corchetes para acceder al contenido de la memoria aunque se pueden usar paréntesis usando la opción "-p" por línea de comandos pero personalmente no lo recomiendo porque crea confusión con las operaciones lógicas y las indirecciones de memoria:

```
<tab> ld a, b          ; Bien
<tab> LD A, B          ; Bien
ld a, b                ; Mal, no al principio de una línea
<tab> Ld A, B          ; Mal, o mayúsculas o minúsculas
<tab> LD A, b          ; Bien, pero mejor no mezclar mayúsculas y minúsculas
<tab> LD A, [$C000]    ; Bien
<tab> ld a, ($C000)    ; Bien con la opción "-p"
<tab> ld a, (('Z'+1)/2); Con la opción "-p" equivale a ld a, [('Z'+1)/2]
<tab> ld a, (('Z'+1)/2); Sin la opción "-p" equivale a ld a, ('Z'+1)/2
```

Los registros se pueden usar en mayúsculas o en minúsculas pero los registros **ix** e **iy** sólo tienen las siguientes combinaciones:

```
ix, ixl, ixh, IX, IXL, IXH
iy, iyl, iyh, IY, IYL, IYH
```

Etiquetas: Representan direcciones de memoria, variables y constantes, distinguen entre mayúsculas y minúsculas y pueden ser de cualquier longitud razonable pero **siempre debe comenzar al principio de una línea** y es opcional ser seguidas por : (dos puntos). Las etiquetas deben comenzar con una letra o un _ (subrayado) nunca con un número, en los siguientes caracteres se pueden elegir entre letras, números y _ (subrayado) y .(punto).
Los '.' también se utilizan para separar nombres de módulos y usar etiquetas locales. Cuando una etiqueta no empieza con un . (Punto), es una etiqueta no local, cuando una etiqueta empieza con un . (Punto), es local a la etiqueta no local anterior.

```
label           ; Etiqueta no local.
<tab><Código>   ; El código debe empezar tras un tabulador o espacios.
.local          ; Etiqueta local = label.local
<tab><Código>
<tab>label2:    ; Mal, las etiquetas deben ir al principio de la línea
```

En general el formato de una línea de código es el siguiente:

```
[etiqueta[:]] [repetición] [operación [operadores]] [; comentario ]
```

```
MulBy8: [3]     add   a, a       ; a=a*8
```

Mapas de Almacenamiento: Son un tipo de etiquetas ideadas especialmente para alojar datos en RAM, paso a explicarlos y su equivalente con la directiva **ORG** y **DS**:

```
<tab> MAP $C000              ; ORG $C000
etiqueta1   # 1              ; etiqueta1:     DS 1
etiqueta2   # 256            ; etiqueta2:     DS 256
estructura  # 0              ; estructura:
miembro1    # 1              ; miembro1:      DS 1
miembro2    # 2              ; miembro2:      DS 2
miembro3    # 1              ; miembro3:      DS 1
<tab> MAP @ miembro1         ; ORG miembro1
miembro4    # 4              ; miembro4       DS 4
<tab> ENDMAP
```

No usar la directiva **ORG**, continúa sólo por compatibilidad y debe usarse **MAP** y **CODE** en sustitución.

Si una etiqueta **no local** comienza con un **!** (Signo de exclamación), la etiqueta no local no se utilizará como tal en el sistema de etiquetas locales, ver ejemplo:

```
label1           ; label1
<tab> <Código>
!label2          ; label2
<tab> <Código>
.local           ; label1.local
<tab> <Código>
```

Módulos: Los módulos hacen locales todas las etiquetas:

```
<tab> MODULE vdp
label            ; vdp.label
<tab> <Código>
.local           ; vdp.label.local
<tab> <Código>
<tab> ENDMODULE vdp
```

Se puede acceder a una etiqueta local de un módulo asi: módulo.etiqueta

```
<tab> CALL vdp.label
```

Utilice un @ para omitir el uso de una etiqueta local dentro de un módulo:

```
<tab> MODULE main      ; <TAB> obligatorio
label1           ; main.label1
<tab> <Código>
@label2          ; label2 “no local”
<tab> <Código>
.local           ; main.label2.local
<tab> <Código>
<tab> ENDMODULE main
<tab> CALL  label2     ; Sin problemas es no local
```

Si se combina !@ se omite el módulo y las etiquetas locales:

```
<tab> MODULE main
label1           ; main.label1
<tab> <Código>
!@label2         ; label2
<tab> <Código>
.local           ; main.label1.local
<tab> <Código>
<tab> ENDMODULE main
```

Unos cuantos ejemplos de etiquetas:

```
label            ; label
.local           ; label.local

label1           ; label1
.local           ; label1.local
label2           ; label2
.local           ; label2.local

label1           ; label1
!label2          ; label2
.local           ; label1.local

<tab> MODULE main
label1           ; main.label1
@label2          ; label2
<tab> ENDMODULE main

<tab> MODULE main
label            ; main.label
.local           ; main.label.local
<tab> ENDMODULE main

<tab> MODULE main
label1           ; main.label1
@label2          ; label2
.local           ; main.label2.local
<tab> ENDMODULE main

<tab> MODULE main
label1           ; main.label1
!@label2         ; label2
.local           ; main.label1.local
<tab> ENDMODULE main
```

Asignamiento de valores a etiquetas (variables):

```
label1 = 13           ; etiqueta con el valor 13 no modificable
label2 := 13          ; etiqueta con el valor 13 modificable
label2 := label1+1    ; label2 := 14
DEFINE valor 14
DEFINE texto "Hola Mundo"

<tab> LD A, label1    ; Igual a LD A, 13
IFDEF valor
     LD A, valor
ENDIF
```

Ya no es necesario usar la directiva **EQU** y sólo continúa por compatibilidad.

Operadores: En general se pueden usar operaciones matemáticas con la misma sintaxis del C++. Los siguientes operadores pueden ser usados en las expresiones:

```
$     $          Dirección actual (ORG)
#     #          Página actual    (PAGE)
:     :label     Página de la etiqueta
::    ::page     La dirección más alta en la página

()    (x+1)/2    Cambia el orden de procedencia

low   low  x     los 8 bits bajos de un valor de 16 bit
high  high x     los 8 bits altos de un valor de 16 bit
not   not  x     no lógico

**    x**y       potencia

*     x*y        multiplicación
/     x/y        división
%     x%y        modulo
mod   x mod y    modulo

+     x+y        suma
-     x-y        resta
-x               cambio de signo x = x * -1

<<    x<<y       desplazamiento izquierda con signo
>>    x>>y       desplazamiento derecha   con signo
>>>   x>>>y      desplazamiento derecha   sin signo
shl   x shl y    desplazamiento izquierda con signo
shr   x shr y    desplazamiento derecha   con signo

<?    x<?y       mínimo
>?    x>?y       máximo

<     x<y        menor que
>     x>y        mayor que
<=    x<=y       igual o menor que
>=    x>=y       igual o mayor que

=     x=y        igual
==    x==y       igual
!=    x!=y       no igual

~     ~x         complemento

!     !x         operador lógico not

&     x&y        operador lógico and
and   x and y    operador lógico and

^     x^y        operador lógico xor
xor   x xor y    operador lógico xor

|     x|y        operador lógico or
or    x or y     operador lógico or

:      x:y       x*256+y
```

Ejemplos:

```
      3<<2                  ; 3 * 2 * 2 = 12
      1+1                   ; 1 + 1 = 2
      high (8000h+(3&1))    ; high(8003h) = 80h
      3>?5                  ; máximo = 5
```

Constantes numéricas:

```
      12     decimal
      12d    decimal
      0ch    hexadecimal
      0xc    hexadecimal
      $c     hexadecimal
      #c     hexadecimal
      1100b  binario
      %1100  binario
      14q    octal
      14o    octal
```

Constantes de caracteres: Se pueden definir con coma simple o con doble coma según convenga, pero con coma doble se aceptan secuencias de escape:

```
      'A'         65
      "A"         65
      "\\"        92
      "\?"        63
      "\'"        39
      "\""        34
      "\A"        7
      "\B"        8
      "\D"        127
      "\E"        27
      "\F"        12
      "\N"        10    Fin de línea
      "\R"        13    Retorno de carro
      "\T"        9     Tabulador horizontal
      "\V"        11    Tabulador vertical
```

Constantes de cadena: Se pueden definir con coma simple o con doble coma según convenga:

```
       "it's fun to be here"
       'how much "fun" do you want?'
       "hoppa\n"
```

Definición de datos:

```
      DB    ; 8-bit constantes y cadenas
      BYTE  ; 8-bit constantes y cadenas
      DW    ; 16-bit constantes
      WORD  ; 16-bit constantes
      DD    ; 32-bit constantes
      DWORD ; 32-bit constantes
      DS longitud [,<valor_por_defecto>] ; Bloques de datos
      DC    ; cadena con el último carácter con el bit 7 a 1
      DZ    ; cadena terminada en un cero
      DT    ; 24-bit constantes
      DEFB  ; 8-bit constantes y cadenas
      DEFM  ; 8-bit constantes y cadenas
      DEFW  ; 16-bit constantes
      DEFD  ; 32-bit constantes
```

Ejemplos:

```
cabecera:  DB $41, $42
texto:     DB "Hola mundo", 0
pagina3:   DW $C000
buffer:    DS 256, 0
```

Estructuras de datos: Permite agrupar varias etiquetas que compartan una identidad común en grupos lógicos o estructuras. Primero se define la estructura y después se declara usándola como cualquier otro tipo de dato:

```
STRUCT scolor          ; scolor = 3     ; La longitud en bytes de la estructura
red    DB 127          ; scolor.red   = offsets 0
green  DB 32           ; scolor.green = offsets 1
blue   DB 64           ; scolor.blue  = offsets 2
ENDSTRUCT

Color_array:     [10]  scolor           ; Define 10 estructuras scolor en ROM
Color_array:     #     scolor * 10      ; Define 10 estructuras scolor en RAM
```

UPDATE actualiza el archivo de salida actual con el nuevo nombre especificado. **OUTPUT** define un nuevo archivo de salida con todo lo compilado anteriormente, si está dentro de un módulo el archivo resultante será sólo el código de dicho módulo, si no, será el total del código compilado. Pueden crearse varios archivos de salida. Ten en cuenta que si usas **OUTPUT** se ignorará el archivo de salida introducido por la línea de comandos.

```
<tab> UPDATE nombre_archivo_salida.ext
<tab> OUTPUT nuevo_archivo_salida.ext
```

Páginas: Para poder usar las páginas con código primero hay que definirlas con **DEFPAGE** y luego activarlas con **PAGE**. La página de RAM usada para variables nunca hay que definirla con **DEFPAGE**, hay que usar mapas **MAP** de memoria que sustituyen a **ORG** $C000 o. Si se quiere empezar en una dirección concreta dentro de una página de código hay que usar **CODE** @ dirección. Las páginas son lógicas así que se pueden definir hasta 256 y con tamaños superiores o inferiores a 16K dando una gran libertad de uso:

```
<tab> DEFPAGE numero_de_pagina [, origen [, tamaño]]

<tab> PAGE numero_de_pagina        ; Activa la página lógica indicada.

<tab> CODE [@ direccion]           ; Para posicionarse en una dirección
<tab> ORG direccion                ; Mejor no usar, usar CODE
ETIQUETA EQU DATO                  ; Usar ETIQUETA = DATO u DEFINE ETIQUETA DATO

<tab> MAP [@] dirección_RAM        ; Para definir variables en RAM
```

NOTA MUY IMPORTANTES:

En el compilador **sjasmpg** a diferencia de otros como **AsMSX** no se definen cabeceras de ningún tipo (ROM, BIN, COM) ni se activan automáticamente las páginas físicas (SLOTS y SUBSLOTS), estos menesteres hay que hacerlos manualmente poniendo el código en ensamblador correspondiente. Hay mucha información en la red sobre el estándar MSX, de todos modos con esta versión del **sjasmpg** también se suministran librerías auxiliares y ejemplos de roms de 32K y 48K. Para roms de 16K no es necesario activar físicamente ninguna página, quedando la típica configuración PAGE 0 = 16K ROM BIOS, PAGE 1 = 16K ROM BASIC, PAGE 2 = Nuestra ROM de 16K y PAGE 3 = RAM 16K (MAP $C000) para usar en variables menos la usada por el sistema de la dirección $F380 a la $FFFF.

Archivos: Si bien se puede hacer un programa en un único archivo todo junto, es muy recomendable dividir un proyecto en módulos específicos de código y en datos, para esto tenemos las directivas **INCLUDE** para insertar código a compilar desde otro archivo y **INCBIN** que inserta directamente sin procesar el contenido de un archivo:

```
INCLUDE    "archivo_de_código"    ; Usa la ruta actual o la especificada.
INCLUDE    <archivo_de_código>    ; Usa la ruta definida por defecto.
<tab>INCDIR "ruta_por_defecto"    ; Define la ruta por defecto.
<tab>INCBIN "archivo_de_datos" [,inicio, longitud]
<tab>INCBIN <archivo_de_datos> [,inicio, longitud]
```

Ensamblado Condicional y Macros:

```
IF <condición>
<tab> <Código>
ELSEIF <condición>
<tab> <Código>
ELSE
<tab> <Código>
ENDIF

DEFINE nombre reemplazamiento

DEFINE nombre(argumento) reemplazamiento [\]

MACRO nombre
<tab> <Código>
ENDMACRO

MACRO nombre arg1 arg2
<tab> <Código arg1>
<tab> <Código arg2>
ENDMACRO
```

Ejemplos:

```
DEFINE LOAD(REG) LD A, REG

MACRO BREAKPOINT
<tab> DB    $40, $18, $00
ENDMACRO

MACRO LOAD REG
<tab> LD    A, REG
ENDMACRO
```

PROCEDIMIENTO DE COMPILACIÓN:

La primera pasada es la pasada del preprocesador. El preprocesador hace lo siguiente en este orden:

- Eliminar todos los comentarios
- Concatenar líneas divididas con \
- Procesa los comandos **"define"**
- Reemplazar macros de texto y funciones de macros
- Borra los espacios en blanco
- Procesar otros comandos de preprocesador como **if**, **while** y **struct**
- Corta múltiples líneas de instrucciones en pedazos
- Expande los macros de procedimiento

En la primera pasada sólo puede utilizar (o referencia) etiquetas que se definen antes de ser utilizarlas.

En el segundo paso, SjasmPG trata de averiguar qué valores deben tener todas las etiquetas y en qué orden debe colocar todas las partes del código. No es tan difícil hacer imposible que SjasmPG haga esto, y hacer que SjasmPG falle.

Cuando todas las etiquetas tienen los valores correctos SjasmPG genera la salida en la tercera pasada.

Después de eso, SjasmPG revisa todo el código de nuevo para generar el archivo de listado y para ver si puede generar más mensajes de error.

SINTAXIS GENERAL:

[etiqueta[:]] [repetición] [operación [operadores]] [; comentario]

Etiquetas y Directivas del Preprocesador sin espacios ni tabuladores, siempre al principio de una nueva línea. El resto de directivas e instrucciones siempre detrás de una etiqueta o un tabulador o espacio, según corresponda.

Las Directivas del Preporcesador son pocas y es fácil recordarlas, son las siguientes:

if, **ifdef**, **ifndef** y derivados.
else, **elseifdef**, **elseifndef** y derivados
endif

assign, **tostr**, **strlen**, **substr**, **rotate**

while, **while**, **endwhile**, **repeat**, **repeat**, **endrepeat**, **break**, **continue**

struct, **endstruct**, **ends**

define, **idefine**, **xdefine**, **xidefine**, **undef**

macro, **macro.**, **imacro**, **imacro.**, **exitmacro**, **xexitmacro**, **endmacro**, **endm**

include

PREPROCESADOR: (Pueden y deben empezar al principio de una línea)

if <expresión> si la expresión se evalúa con un valor distinto de cero.
ifdef <identificador> si el identificador es una macro de texto.
ifndef <identificador> si el identificador no es una macro de texto.
ifb <argumento> si el argumento está vacío.
ifnb <argumento> si el argumento no está vacío.
ifnum <argumento> si el argumento puede ser un número.
ifnnum <argumento> si el argumento no puede ser un número.
ifstr <argumento> si el argumento puede ser una cadena.
ifnstr <argumento> si el argumento no puede ser una cadena.
ifexists <filename> si el archivo existe.
ifnexists <filename> si el archivo no existe.
ifid <argumento> si el argumento puede ser una etiqueta.
ifnid <argumento> si el argumento no puede ser una etiqueta.
ifdif <argumento1>, <argumento2> si argumento1 es diferente de argumento2.
ifdifi <argumento1>, <argumento2> ifdif no sensible a mayúsculas y minúsculas.
ifidn <argumento1>, <argumento2> si argumento1 es idéntico a argumento2.
ifidni <argumento1>, <argumento2> ifidn no sensible a mayúsculas y minúsculas.
ifin <argumento>, <lista> si el argumento existe en la lista.
ifini <argumento>, <lista> ifin no sensible a mayúsculas y minúsculas.
else
elseifdef <identificador> si el identificador es una macro de texto.
elseifndef <identificador> si el identificador no es una macro de texto.
elseifb <argumento> si el argumento está vacío.
elseifnb <argumento> si el argumento no está vacío.
elseifnum <argumento> si el argumento puede ser un número.
elseifnnum <argumento> si el argumento no puede ser un número.
elseifstr <argumento> si el argumento puede ser una cadena.
elseifnstr <argumento> si el argumento no puede ser una cadena.
elseifexists <filename> si el archivo existe.
elseifnexists <filename> si el archivo no existe.
elseifid <argumento> si el argumento puede ser una etiqueta.
elseifnid <argumento> si el argumento no puede ser una etiqueta.
elseifdif <argumento1>, <argumento2> si argumento1 es diferente de argumento2.
elseifdifi <argumento1>, <argumento2> ifdif no sensible a mayúsculas y minúsculas
elseifidn <argumento1>, <argumento2> si argumento1 es idéntico a argumento2.
elseifidni <argumento1>, <argumento2> ifidn no sensible a mayúsculas y minúsculas
elseifin <argumento>, <lista> si el argumento existe en la lista.
elseifini <argumento>, <lista> ifin no sensible a mayúsculas y minúsculas.
endif

assign
tostr
strlen
substr
rotate

while <condición> Mientras <condicion>
while. Igual que while
endwhile Fin de while
repeat <cantidad> Repite <cantidad>
repeat. Igual que repeat
endrepeat Fin de repeat
break Rompe un while o repeat.
continue Continua un while o repeat.

struct name [, initial offset, alignment]
endstruct Fin de struct
ends Igual que endstruct

define
idefine
xdefine

xidefine
undef

macro
macro. Igual que macro
imacro
imacro. Igual que imacro
exitmacro
xexitmacro
endmacro
endm Igual que endmacro

include "archivo_de_código" Usa la ruta actual o la especificada.
include <archivo_de_código> Usa la ruta definida por defecto.

PROCESADOR DE DATOS:
(**¡Ojo!** Deben empezar después de una etiqueta, tabulador o un espacio)

```
[label] [[repeatcount]] COMMAND [expr][,expr]... [comment]
label ## alignment value            ; align the following field
label #  length                     ; define a field of the given length
```

byte 8-bit constants and strings
db 8-bit constants and strings
defb 8-bit constants and strings
defm 8-bit constants and strings

dc strings; every last character of a string will have bit 7 set
dz strings; each string will be zero terminated

word 16-bit constants
defw 16-bit constants
dw 16-bit constants

dt 24-bit constants

dword 32-bit constants
defd 32-bit constants
dd 32-bit constants

[label] DS [cantidad] [valor_inicial]
ds
defs
block

[label] [repeatcount] ABYTE [offset] [expr][,expr]... [comment]
abyte
abytec
abytez

asc
ascc
ascz
ascmap
ascmap.reset
ascmap.clear

DIRECTIVAS: (**¡Ojo!** Siempre deben empezar después de un tabulador o un espacio)

code [?], [@ address], [# alignment], [PAGE page]
? El código se omite si hay alguna etiqueta no referenciada.
@ dirección El código empieza en la dirección especificada.
@ inicio..fin El código empieza en el rango especificado.
Ejemplo:
code @ 100h..300h, # 64, page 0

align Alinea las direcciones de código.

end Fin del Programa, no se procesa más código, sólo debe haber uno.

module nombre Define un espacio propio u módulo de etiquetas.
endmodule nombre Fin del módulo.

defpage numero_de_pagina [, origen [, tamaño]]
page numero_de_pagina Activa la página seleccionada

map [@] direccion Establece la dirección para etiqetas con #
@ Cambia la dirección del mapa actual (no en sjasm)
mapalign Alinea las etiquetas definidas con #
Igual que mapaling
endmap Fin del map actual

phase direccion Permite compilar en una dirección virtual.
dephase Fin de phase

org direccion Establece la dirección donde se compila el código.
Esta directiva está por compatibilidad, usar **CODE**

etiqueta **equ** valor Asigna un valor numérico a una etiqueta.
Esta directiva está por compatibilidad, usar **=** o **DEFINE**
incdir "ruta_por_defecto"

incbin "archivo_de_datos" [,inicio, longitud] Usa la ruta especificada.
incbin <archivo_de_datos> [,inicio, longitud] Usa la ruta por defecto.
incbin. Es lo mismo que **incbin**

update archivo_de_salida.ext
output nuevo_archivo_de_salida.ext

assert
error
z80 Define las Instrucciones Z80 (Usado por defecto)
size tamaño_en_Kb Define el tamaño de salida. (No es necesario usarlo)

Nuevas Directivas sólo para SjasmPG:

printstr "Texto" [, cantidad_endl] Imprime una cadena de texto
printdec valor [, ancho] Imprime valor en decimal
printhex valor [, ancho] Imprime valor en hexadecimal
printendl Imprime un final de linea
printstrdec "Texto" [,valor] [,ancho] Imprime STR + [, DEC] [, ancho] + ENDL
printstrhex "Texto" [,valor] [,ancho] Imprime STR + [, HEX] [, ancho] + ENDL

Recomiendo no usar "\n" o "\r\n" en la directiva **printstr**, es mejor el comando opcional que inserta la cantidad deseada de nuevas líneas. También recomiendo usar **printendl** para optimizar la salida en consola y en archivo.

INSTRUCIONES Z80:

(**¡Ojo!** Deben empezar después de una etiqueta un tabulador o un espacio)

adc	Suma con Acarreo
add	Suma
and	And Lógico
bit	El estado de un bit
call	Llama a una función
ccf	Not(CF)
cp	ComPara
cpd	Busca
cpdr	Busca
cpi	Busca
cpir	Busca
cpl	Not(A)
daa	Decimal Adjust Accumulator
dec	DECrementa
di	Desactiva las Interrupciones
djnz	Salto condicional
djnz.	Igual que **djnz**
ei	Activa las Interrupciones
ex	EXchange
exx	EXchange BC,DE,HL<->BC',DE'HL'
halt	Espera a la próxima Interrupción
im	Selecciona el tipo de Interrupción
in	Puerto de Entrada
inc	INCrementa
ind	Puerto de Entrada
indr	Puerto de Entrada
ini	Puerto de Entrada
inir	Puerto de Entrada
jp	Salto Incondicional Absoluto
jp.	Igual que **jp**
jr	Salto Incondicional Relativo
jr.	Igual que **jr**
ld	LoaD
ldd	Copia
lddr	Copia
ldi	Copia
ldir	Copia
mulub	**¿?**
muluw	**¿?**
neg	Complemento
nop	Nada
or	Or Lógico
otdr	Puerto de Salida
otir	Puerto de Salida
out	Puerto de Salida
outd	Puerto de Salida
outi	Puerto de Salida
pop	Pop Mete en la Pila
push	Push Saca de la Pila
res	Resetea un bit a 0.
ret	Retorno de Función Interrupción
reti	Retorno de Interrupción
retn	Retorno de Interrupción(No usado en MSX)
rl	Rotación Izquierda
rla	Rotación Izquierda
rlc	Rotación Izquierda
rlca	Rotación Izquierda
rld	Rotación Izquierda
rr	Rotación Derecha
rra	Rotación Derecha
rrc	Rotación Derecha

rrca	Rotación Derecha
rrd	Rotación Derecha
rst	Llama a una Función de Interrupción.
sbc	Resta con acarreo
scf	CF=1
set	Setea un bit a 1
sla	Desplazamiento Izquierda
sll	Desplazamiento Izquierda
sra	Desplazamiento Derecha
srl	Desplazamiento Derecha
sub	Resta
xor	XOR Lógico

Ejemplo de una ROM de 16K:

```
; Macros:
MACRO BREAKPOINT            ; Para el depurador blueMSX
      LD    B, B
      JR    $+2
ENDMACRO

; Cabecera del cartucho:
      DEFPAGE 1, $8000, $4000
      PAGE  1               ; La página en ROM
      MAP $C000             ; La página en RAM
      CODE  @ $8000
      BYTE  $41, $42        ; Cabecera tipo ROM
      WORD  inicio          ; Punto de inicio del código.
      WORD  0,0,0,0,0,0     ; 12 bytes de cabecera.

; Librerías, Datos y Constantes:
INCLUDE     "librería.asm"
imagen:     INCBIN      "imagen.bin"
ancho = 40
DEFINE alto 25

; Variables en RAM:
      buffer # 256          ; buffer de 256 bytes en RAM


; Inicio del programa:
inicio:

      MODULE main
      <Código>
      ...
      BREAKPOINT            ; Punto de ruptura
      ...
      ENDMODULE main

; Fin de programa
fin_de_rom:
      PRINTSTRHEX "Memoria ROM consumida = $", fin_de_rom - $8000

fin_de_ram # 0
      PRINTSTRHEX "Memoria RAM consumida = $", fin_de_ram - $C000

      ENDMAP
      END
```
