Nerlaska. Ver. 12.12.06

COMO CREAR ROMS DE 32KB PARA MSX CON SDCC

Objetivo

Poder crear ROMS de hasta 32 Kb para MSX usando la herramienta gratuita SDCC.

Requerimientos

- Disponer de un PC Compatible
- Instalar el compilador SDCC (http://sdcc.sourceforge.net/snap.php). Recomiendo bajarse el último SNAP para Windows Binaries, preferiblemente con instalador. Los Snap se actualizan prácticamente cada semana.
- Tener ciertos conocimientos sobre como funciona el Z80 y sobre paginación y uso de la memoria del MSX.
- Disponer de los binarios Hex2Bin y FillFile. (http://www.nerlaska.com/msx/hex2bin.rar) (http://www.nerlaska.com/msx/fillfile.rar)

El compilador SDCC

Este compilador se instala por defecto en una carpeta dentro de "Archivos de Programas". Tiene varios binarios para desempeñar distintas funciones pero el que a nosotros nos interesa es el binario "sdcc.exe".
Por defecto al instalar, se agregará al PATH un acceso a la carpeta de binarios SDCC para que desde el DOS-PROMPT tengamos acceso directo a los mismos.

¿Como es una ROM de MSX?

Básicamente hay que saber que una ROM de MSX tiene una cabecera de 16 bytes y a continuación viene todo el código.

```
41 42 10 40 00 00 00 00 00 00 00 00 00 00 00 00
```

Aquí la tienes en formato hexadecimal. El 41 y el 42 son las letras A B. El "10 40" es la dirección de arranque en nuestro código. El resto de bytes remito a la documentación facilitada por ASCII en su MSX2 Technical Handbook. Por lo que a nosotros respecta esos bytes deben ser 0.

Consideraciones sobre SDCC

Bien, ya casi estamos a punto de generar nuestra primera ROM hecha con SDCC. Sin embargo antes, me gustaría aclara algunas cosas de cómo trabajar con "control" usando SDCC.
Por defecto el SDCC dispone de código Z80 con soporte para las librerías estándar de C como son la stdlib y la stdio. Además de disponer de librerías para el uso de floats y longs (32 bits), multiplicación y división entre otro tipo de cosas.
De todo esto nos vamos a olvidar!!
Es importante, porque si usamos todo esto, además de que nos hinchará el código brutalmente (y 32 kb. Son realmente pocas kb) el problema será que no tendremos un

control de donde se ubica ese código. El SDCC determina por si mismo (nosotros no podemos controlarlo) donde ubica su código de librería.

Vamos a crear nuestra primera ROM

Crearemos un archivo "prueba.c" con el siguiente contenido:
Tranquilos, enseguida comento que diablos es todo esto que hay aquí escrito ☺

```
sfr at 0xA8 g_slotPort;

void main (void)
{
      _asm
            di
            ld sp, (#0xFC4A)
            ei
      _endasm;

      g_slotPort = (g_slotPort & 0xCF) | ((g_slotPort & 0x0C) << 2);

      // A partir de aquí! Nuestro código…
      while (1);
}
```

¿Cómo se compila?

Ponemos el Dos-Prompt: Windows -> Inicio -> Ejecutar -> cmd.exe
Y desde aquí, si el SDCC está correctamente instalado hacemos:

```
sdcc -mz80 --no-std-crt0 --code-loc 0x4010 --data-loc 0xC000 prueba.c
```

Esto nos genera los siguientes archivos:

```
Prueba.o
Prueba.sym
Prueba.map
Prueba.asm
Prueba.ihx
Prueba.lst
Prueba.lnk
```

De todos estos, los interesantes son el "prueba.o" por si queremos hacer uso de librerias o compilar varios C. Esto lo veremos más adelante.
El "prueba.map" es del todo interesante, pues te indica las direcciones, etiquetas y memoria empleada por el código generado.

Y uno de los más importantes: El "prueba.ihx" nuestro código en modo hexadecimal. Que ahora con la herramienta Hex2Bin conseguiremos pasar a BIN.

```
hex2bin –e bin prueba.ihx
```

Ahora tendremos un nuevo archivo "prueba.bin". Este es nuestro código ROM a falta de una cosa muy importante!! La cabecera de 16 bytes.

¿Cómo la agregamos?

Muy sencillo. Tenemos por un lado un "rom_header.bin" de 16 bytes que no es más que un fichero con los datos que necesitamos.
Desde el DOS-PROMPT hacemos:

```
copy /b rom_header.bin + /b prueba.bin /b prueba.rom
```

Ahora ya tenemos nuestra ROM creada pero! Todavía hay un problema que subsanar. Necesitamos completar hasta 32 Kb nuestra ROM. Sobre todo si queremos poder usarla en un emulador como pueda ser el BlueMSX (recomendado 100%)

¿Cómo se completa?

Bueno, para ello he creado un programita muy sencillo llamado "FillFile".
Desde el DOS-PROMPT hacemos:

```
fillfile prueba.rom 32768
```

Tras realizar esta operación tendremos un archivo ROM de 32Kb.

¿Qué es todo ese código en C?

Ya hemos llegado al punto de explicar un poco que hemos estado haciendo para generar esa ROM de forma correcta.
Para quien no lo sepa, debería saber que una ROM de 32Kb se mapea a través del SLOT_1(0x4000) y SLOT_2(0x8000) del área direccionable.
Por lo tanto, las direcciones para el código se encuentran entre 0x4000 y 0xBFFF.
Para las variables, se utilizará la RAM que se encuentra en el SLOT_3(0xC000).
Quedando en el SLOT_0 el código de la BIOS.
Hay que tener en cuenta que nuestro programa C también usa la Pila y que esta funciona sobre el SLOT_3, vamos, sobre la RAM.

¿Como arrancamos nuestro programa entonces sabiendo todo esto?

Vamos por partes.

```
sfr at 0xA8 g_slotPort;
```

Esto es una sentencia del compilador que nos permite trabajar con puertos directamente. En este caso creo un acceso al puerto para gestión de slots.

```
void main (void)
{
```

Bueno, ahora creo el punto de entrada de mi ROM. Realmente podría tener cualquier nombre la función, pero por seguir un poco el estándar de C, la seguiré denominando "main". Lo que nos importa es que al compilar, esta función se genere en la posición 0x4010. Que son 16 bytes justo después de nuestra cabecera ROM.

A continuación viene la inicialización de nuestra pila. Para ellos usaremos una variable BIOS que determina la posición de inicio de la pila.

```
	_asm
		di
		ld sp, (#0xFC4A)
		ei
	_endasm;
```

Ahora configuraremos nuestro puerto de slots para que la ROM arranque correctamente independientemente del zócalo usado en nustros MSX. En teoría una ROM es un cartucho y este se puede insertar, generalmente, en uno de los 2 zócalos creados para tal propósito. Hasta aquí bien, el asunto está en que nuestra ROM es de 32 Kb, por lo tanto necesitamos establecer que será el SLOT_1 y SLOT_2 de nuestra área de direccionamiento. El ordenador cuando arranca configura el puerto 0xA8 debidamente mirando si hay algún cartucho insertado. Nosotros lo único que tenemos que hacer es asociar al SLOT_2 el mismo valor que el asociado al SLOT_1.

```
	g_slotPort = (g_slotPort & 0xCF) | ((g_slotPort & 0x0C) << 2);
```

A partir de aquí, ya podemos añadir nuestro propio código.

```
	while (1);
}
```

¿Cómo usar assembler inline?

El SDCC incluye un binario para poder compilar código ensamblador. Aún así podemos incluir código ensamblador directamente en nuestro código C.
Antes ya hemos visto un poco como hacer esto, pero vamos a explicar como poder usar Assembler Inline cómodamente y en mi opinión adecuadamente con el uso del C.

Salvo excepciones como el main, o código crítico que requiera de optimización especial, lo normal es usar assembler inline como interface en funciones.
Por ejemplo:

```
void VDP_FillVRAM (unsigned int begin, unsigned int size, unsigned char value)
{
	begin;
	size;
	value;
_asm
	ld l, 4(ix)
	ld h, 5(ix)
	ld c, 6(ix)
	ld b, 7(ix)
	ld a, 8(ix)
	call 0x0056
_endasm;
}
```

SDCC prepara el registro IX automáticamente para poder tener acceso a los argumentos pasados a la función de forma cómoda. Si quisiéramos poder retornar algún valor, podemos hacerlo de la siguiente manera:

```
unsigned char VDP_ReadVRAM (unsigned int addr)
{
        addr;
_asm
        ld l, 4(ix)
        ld h, 5(ix)
        call 0x004a
        ld h, #0x00
        ld l, a
_endasm;
}
```

SDCC devuelve los valores en el par HL. Sólo en L si se devuelve 1 byte, y en HL si se devuelven 2 bytes.

Anotación:
Habréis notado que están escritos los argumentos pasados en la función al comienzo del código seguidos de punto y coma. Bueno, esto es un apaño para que el compilador no genere WARNINGS por el no uso de los argumentos dentro de la función.
En el caso de funciones que devuelvan valor (como esta última) será inevitable que el compilador genere un WARNING advirtiendo de que no se devuelve ninguno. Esto es mentira, el compilador no se ha dado cuenta de que si lo hago, pero no hay que preocuparse por esto si sabemos lo que estamos haciendo.

¿Cómo trabajar con varios C?

Este punto es importante, porque la gracia de todo esto es tener una buena organización de nuestro código. No es nada interesante tener todo metido en un único C, pero bueno, *sobre gustos no hay nada escrito.*

Veamos:

```
sdcc -mz80 --no-std-crt0 --code-loc 0x4010 --data-loc 0xC000 p1.c p2.c
p3.c p4.c p5.o p6.asm p7.c main.c
```

Esto compilaría y enlazaría los archivos P1, P2, P3, P4, P5, P6, P7 y MAIN.
Como se puede observar, pueden ser de diferentes extensiones. SDCC ya se encarga de saber en que formato están.
Lo importante aquí es que el MAIN sea siempre el último! Porque de no ser así no se colocará en la posición 0x4010 establecida.
Usando este modo de compilar, veréis en el .MAP generado que las direcciones se van asignando de izquierda a derecha hasta llegar al MAIN que se reserva como primera posición.

Anotación:
SDCC permite usar librerías. Es una forma de recopilar varios archivos objeto .O en un solo archivo .LIB.
Mi experiencia me ha dicho que no me gusta esto, porque al usar .LIB dejas de controlar la asignación de direcciones, ya que SDCC se salta el orden que antes hemos visto cuando usa librerías. Es importante tener el control de las direcciones, que luego necesitamos agregar o quitar código y si no tenemos control sobre esto, puede ser una auténtica locura!