Lenguaje ensamblador en el Sinclair QL, mis primeros pasos.

1ª Parte

Por afx, marzo 2009

Preámbulo

El inicio en el aprendizaje de un nuevo lenguaje de programación suele ser algo desconcertante en las etapas iniciales. En muchas ocasiones, y por supuesto dependiendo de tus conocimientos previos, tu mente se debe adaptar a pensar de un modo diferente, debes aprender un montón de palabras reservadas nuevas, debes tener conocimientos del entorno de programación, del compilador, del sistema operativo sobre el que te desenvuelves, etc. Es frecuente también que no encuentres la documentación adecuada adaptada a tu nivel, o que los ejemplos disponibles requieran de unos conocimientos previos elevados. Estas cosas hacen que frecuentemente tiremos la toalla incluso antes de comenzar.

Todo esto es espacialmente cierto si el lenguaje del que estamos hablando es a bajo nivel tal como el lenguaje ensamblador del Motorola 68000, y además en una plataforma extremadamente rara en nuestros días como es el Sinclair QL y su sistema operativo QDOS.

La mejor manera de superar estas dificultades iniciales es a base del estudio y prácticas con ejemplos muy simples de pequeños programas "funcionales", es decir que se puedan compilar y ejecutar ¡sin errores! desconcertantes (sobre todo para no perder la auto-confianza). Debemos empezar con lo simple y poco a poco escalar en dificultad. Después de todo, como se dice frecuentemente, la mejor manera de aprender a programa es ... ¡copiar! ... (bueno, mejor dicho, tomar prestado) el código que ha han escrito otros.

El presente artículo recoge esos "primeros pasos" que he dado a la hora de enfrentarme a este lenguaje y tiene la única pretensión de facilitar un poco la tarea a aquellos aficionados al Sinclair QL que tengan interés en adentrarse en el apasionante mundo de la programación a bajo nivel. Después de todo, esto es otra forma de "dar vida" y "desempolvar" nuestras viejas máquinas, sobre todo para aquellas personas entusiastas de la retro-informática que, como yo, no sean muy dados a los videojuegos. (¿Acaso no es más divertido programar estas viejas máquinas que jugar con ellas?).

El presente artículo no pretende ser un tutorial ni un manual del lenguaje ensamblador o del QDOS ,

para ello ya existe documentación que nos explica con todo lujo de detalles tanto el lenguaje ensamblador del Motorola 68000 como del sistema operativo del Sinclair QL (ver la sección de manuales y artículos en www.sinclairql.es).

La mayor parte del contenido de este artículo y los ejemplos que detallo no son de creación propia (¡qué más quisiera yo!), por el contrario, es material que he rescatado de distintas fuentes, entre ellas:

- Foro www.sinclaiql.es
- Revista Qlave
- Manual del *Assembler Development Kit* de Metacomco
- 68000, 68010 /68020 Arquitectura y programación en ensamblador. Stan Kelly-Bootle y Bob. Fowler. (Anaya Multimedia)
- Sinclair QL Programación Avanzada, Adrian Dickens.

A todo este material se puede acceder a través de la web http://www.sinclairql.es.

Se acompaña a este artículo, un fichero zip con el código fuente de los ejemplos analizados, listos para ser ensamblados con el "Assembler Developtment Kit" de la casa Metacomco. Los ejemplos han sido probados en un QL real con ampliación de disquetera y en el emulador Q-Emulator (¡ahora es "free" la emulación básica!). Además de todo esto podrías necesitar un editor de textos para elaborar tus propios programas en ensamblador o modificar los que aquí se proponen, QED es un buen editor, compacto y rápido).

Los recursos mencionados los puedes encontrar en:

- http://www.bytemaniacos.com/sinclairql/lenguajes.htm (ensamblador de Metacomco)
- http://www.speccy.org/sinclairql/manuales.htm (documentación y bibliografía citada)
- http://www.speccy.org/sinclairql/archivo/docs/docs.htm (manuales Metacomco y M68000)
- http://www.dilwyn.uk6.net/editview/index.html (editores)
- http://www.terdina.net/ql/q-emulator.html (emulador Q-emulator)

Bueno, basta de preámbulos, empecemos a dar esos primeros pasos.

El primer paso, "hola mundo"

"Hola mundo" se ha convertido en un término muy familiar par cualquier persona aficionada o profesional en la programación de ordenadores. Miren si no la definición que nos propone la Wikipedia de ésta expresión:

> *"En informática, un programa Hola mundo (o Hello World, en inglés) es el que imprime el texto «¡Hola, mundo!» en un dispositivo de visualización (generalmente una pantalla de monitor). Se suele usar como introducción al estudio de un lenguaje de programación, siendo un primer ejercicio típico.*
>
> *El Hola Mundo se caracteriza por su sencillez, especialmente cuando se utiliza en*

> *programas de línea de comandos. En interfaces gráficas este programa suele tener una complejidad mayor.*
>
> *Un programa Hola Mundo puede ser útil como prueba de configuración para asegurar que el compilador, que el entorno de desarrollo, y que el entorno de ejecución están instalados correctamente y funcionando. Configurar un conjunto de herramientas básicas completo desde cero hasta el punto donde hasta los programas triviales puedan ser compilados y ejecutados, puede involucrar una cantidad de trabajo sustancial. Por esta razón, generalmente es usado un programa muy simple para probar un conjunto de herramientas nuevo." (www.wikipedia.org).*

Cualquier comentario adicional a esta descripción sobra (¡ni Cervantes lo redactaría tan bien!). Resulta también muy curioso la gran cantidad de lenguajes en los que se ofrece el código "Hola mundo" en Wikipedia, pero si lo miráis veréis que al menos ¡falta uno!, el "hola mundo" en ensamblador 68000. Pues pongamos remedio a eso, a continuación va ese primero programa extraído del foro de www.sinclaiql.es (gracias *Zerover*).

Listado 1, Hola_asm.

```
*
* Códigos de llamadas al QDOS
*
MT_FRJOB EQU $05
IO_SSTRG EQU $07
*
* Programa principal
*
         LEA.L     SALUDO,A3       apuntamos a la frase a escribir con A3
         MOVE.W    (A3),D2         numero de octetos a mandar
         MOVEQ.L   #-1,D3          periodo de espera infinito
         MOVEA.L   #$00010001,A0   identificador del canal #1
         LEA.L     2(A3),A1        apuntar a la cadena de octetos
         MOVEQ.L   #IO_SSTRG,D0    enviar la cadena de octetos
         TRAP #3                   Traps de E/S

         MOVEQ.L   #-1,D1          tarea actual
         MOVEQ.L   #0,D3           ningún error
         MOVEQ.L   #MT_FRJOB,D0    forzar el final de la tarea
*                                  (Force Release JOB)
         TRAP #1                   Traps de gestión de procesos

*
* En caso de ser llamado desde el SuperBASIC (CALL)
         MOVEQ     #0,D0           poner retorno de error a 0 para SuperBasic
         RTS                       vuelve a SuperBASIC
*
* Sección de datos
*
SALUDO   DC.W 11                 las cadenas precedidas de DW longitud
         DC.B 'Hola mundo.'
```

```
        END
```

Algunos comentarios al programa:

El código comienza definiendo una serie de constantes con EQU, muy cómodas para no estar recordando valores concretos, cosa que no forma parte del lenguaje máquina del 68000 propiamente dicho sino que son directivas del ensamblador. Por ejemplo IO_SSTRG EQU $07 al principio del programa provoca que el ensamblador sustituya $07 cada vez que encuentre IO_SSTRG. El $ indica que el numero es expresado en hexadeciamal (y no significa dinero ☹).

A continuación viene la secuencia principal del programa donde vemos muy pocas instrucciones realmente diferentes (MOVE, LEA, MOVEQ) que llevan aparejada un punto y una letra. A esta letra se le llama código de tamaño del dato, porque indica cuantos bits de origen (fuente) y del destino están involucradas en la operación. Las reglas son muy simples:

L *significa doble palabra: opera sobre 32 bits.*
W *significa palabra: opera sobre 16 bits.*
B *significa byte: opera sobre 8 bits inferiores.*

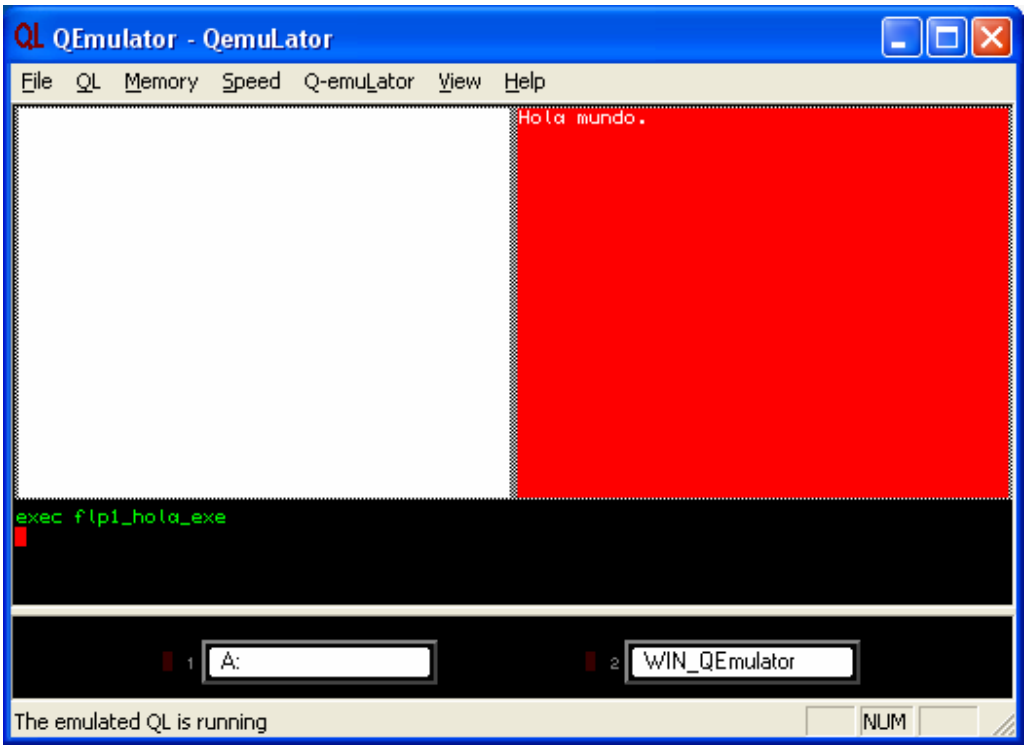
Estas son las descripciones breves de algunas instrucciones empleadas en nuestro primer programa.

LEA	*(Load Effective Address), coloca una dirección de memoria especificada en un registro de direcciones (An)*
MOVE	*mueve el contenido de la fuente al destino.*
MOVEQ	*(MOVE Quick) mueve un dato de 8 bits con signo de forma rápida al destino. El dato especificado que va a ser movido es extendido a 32 bits antes de ser movido al destino.*
MOVEA	*(MOVE Address) mueve el contenido desde el origen a un registro de direcciones.*
TRAP	*el procesador inicia una excepción. Se ejecuta una llamada al sistema.*

Vemos también que en algunas instrucciones se emplean los registros de direcciones entre paréntesis, (A3) por ejemplo. Esto significa que estamos tratando con el contenido de la dirección a la que apunta el registro, en nuestro caso podemos leerlo como el contenido de la posición de memoria a la que apunta A3. 2(A3) significa que tenemos que sumar dos bytes más a la posición apuntada por A3 y tomar el valor de esa posición de memoria.

Finalmente el código finaliza con la sección de datos. En una zona de memoria etiquetada como SALUDO se deposita una palabra (16 bits) con el número 11 y a continuación una secuencia de 11 caracteres que ocupan un byte cada uno de ellos.

Bien, todo esto simplemente para ver en pantalla lo siguiente.



Paso 2, el ensamblado con "Assembler Development Kit" de Metacomco

En el paso anterior vimos el código fuente de nuestro primer programa y el resultado final tras ejecutar el código objeto del programa en un emulador del Sinclair QL, pero ¿cómo pasamos del código fuente al código ejecutable?, … pues en nuestro caso ensamblando el programa fuente con el "Assembler Development Kit" de la casa Metacomco (la misma que fabricó el sistema operativo del Amiga).

Para ejecutar el compilador es preciso teclear "exec flp1_asm" (yo he renombrado asm_metacomco por asm del paquete citado). Al iniciarse el programa se nos hace las siguientes preguntas:

En primer lugar debemos indicar cuál es el nombre del fichero que anteriormente salvamos y que contiene el programa. Después debemos indicar si deseamos listado del programa ensamblado, si pulsamos ENTER se asume que no; si hemos respondido 'Y' entonces de nos pregunta el nombre del fichero al que dirige el listado, se pulsa ENTER se dirige a la pantalla, podemos redirigir también a ser1, flp2_ ... En este listado aparece las líneas que contiene algún error con una E por delante y con signos = las partes que se refieran a macros.

Posteriormente se nos pregunta por el nombre del fichero que contendrá el código objeto. Si tecleamos ENTER no se genera código y el compilado sólo nos servirá para comprobar si hemos

cometido errores en la redacción del programa.

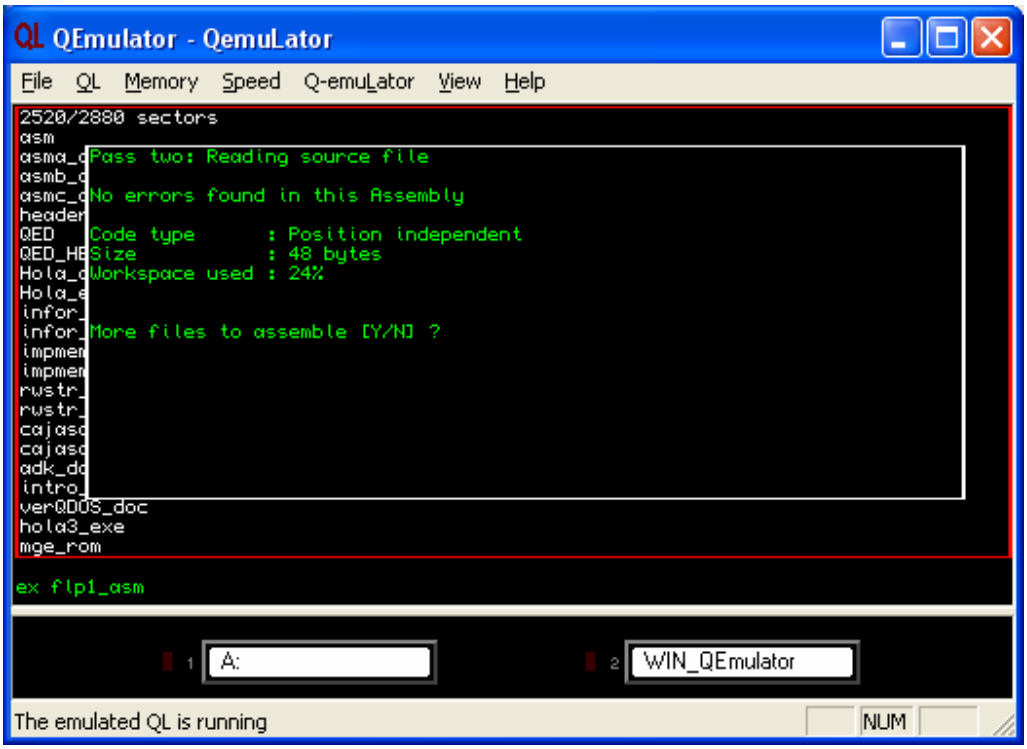
Se nos pregunta también por el espacio de trabajo del programa, si se pulsa ENTER se toma el espacio reservado por defecto. De igual forma se nos interroga sobre la alteración del tamaño de la ventana de forma que un ENTER o 'N' implican dejarla igual. La alteración de tamaño es con ALT+cursores y su situación se controla por los cursores sin ALT.

En las primeras versiones los programas venían instalados para trabajar en el mdv1_ y los ficheros a compilar en el mdv2_, esto era alterable por el fichero install de forma que aquellos que dispongan de una ampliación de memoria puedan cargar los programas en disco ram y dirigir los ficheros compilados ahí, luego bastará hacer un copy a un mdv_ o disco del compilado definitivo. En nuestros ejemplos usaremos una versión adaptada para ser ejecutada en flp1_

Dependiendo del fichero que hayamos deseado compilar se nos generará un código independiente de la dirección de carga o no, así si no hemos incluido ORG por lo general será independiente y ejecutable vía EXEC que suele ser lo deseable, en caso contrario el programa se ejecuta con CALL tras cargarlo con LBYTES. En nuestros ejemplos cargaremos siempre los programas directamente desde el QDOS con EXEC.

Al final se nos pregunta si deseamos compilar más programas, debemos responder 'Y' o 'N'.

A continuación vemos algunas pantallas con el ensamblador en acción, su uso es extremadamente fácil.



Paso 3, mostrar la versión del QDOS

Veamos nuestro segundo ejemplo. Éste pequeño programa está extraído de uno de los primeros números de la revista Qlave. Veremos cómo interrogar a un TRAP del sistema para obtener la versión del QDOS que se está ejecutando en nuestro sistema. Ya en un artículo previo en www.sinclairql.es veíamos que el comando PRINT VER$ del SuperBasic lo que nos da en realidad es la versión del SuperBASIC en sí mismo y no la versión del QDOS (que son dos cosas diferentes). Podría ser necesario conocer esto ya que existen versiones antiguas que no aceptan algunas "llamadas" ya implantadas en las nuevas versiones, de esta forma se puede garantizar el correcto funcionamiento de un programa independientemente de la versión del QDOS instalada. Para saber la versión del QDOS debemos emplear una rutina en lenguaje ensamblador como la detallada en el del listado 2.

Algunos comentarios del autor del ejemplo (Isidro Asin, revista Qlave número 1, volumen 1).

"El programa tiene cuatro bloques básicos, estos cuatro bloques son etiquetas de llamadas al QDOS, macros definidas, el código ejecutable en sí, zona de memoria de datos.

El primer bloque se refiere al valor que se debe pasar en el registro de datos D0 al realizar el TRAP (orden en ensamblador que se refiere a una excepción y que es la forma habitual de trabajo del QDOS) correspondiente. He usado dos Macros, algo bastante recomendable si se dispone de un

ensamblador que lo permita. La primera nos sirve para pasar las llamadas al QDOS (el parámetro en D0 y el número de TRAP que corresponda). La segunda se encarga de finalizar el programa y devolvernos al BASIC.

El programa comienza con una llamada para obtener la información de la versión, esto se hace con el TRAP 1 y MT_INF en D0, el resultado se saca en D2 que guardamos en la zona de memoria reservada para ello, la zona etiquetada como MENSAJE donde con 4 bytes habrá bastado en lugar de 10.

Ahora para sacar estos datos podemos hacerlo por pantalla o por impresora, nosotros lo haremos por pantalla de ahí que DISPOSITIVO sea 'SCR_'. Abrimos un canal para efectuar la salida, para ello usamos un TRAP 2 con IO_OPEN en D0, en D1 el número de job que se trate (en este caso -1 para indicar el actual), en D3 un 2 para indicar que es nuevo, y A0 apunto al DISPOSITIVO, que aunque aquí sea SCR_ podríamos colocar SER1 para impresora por ejemplo.

Para sacar los datos se usa un TRAP 3 con IO_STRING en D0, hay que especificar el número de octetos a mandar en D2 y el tiempo de espera en D3, A1 debe de apuntar a la zona de memoria que contiene los octetos a mandar.

Para finalizar se invoca la Macro FINAL que cierra el canal abierto con un TRAP 2 conteniendo IO_CLOSE en D0. Posteriormente forzamos la clausura del job y como éste es activo lo hacemos con un TRAP 1 conteniendo D0 MT_FRJOB, en D1 está el número de job y en D3 el código de error."

Listado 2, Infor_asm.

```
*
* Códigos de llamadas al QDOS
*
MT_INF      EQU     0           Obtener la versión del QDOS
IO_OPEN     EQU     1           Abrir canales
IO_CLOSE    EQU     2           Cerrar canales
IO_SSTRG    EQU     7           Presenta datos por un canal de salida
MT_FRJOB    EQU     5           Cancela el job
*
* Macros
*
QDOS        MACRO               Macro de asignación de Traps
            MOVEQ   #\1,D0      Parámetro en D0
            TRAP    #\2         Parámetro del Trap
            ENDM
FINAL       MACRO               Macro de cancelación de canales y job
            QDOS    IO_CLOSE,2  Cierra canal
            MOVEQ   #-1,D1      del job actual
            MOVEQ   #0,D3       Código de error
            QDOS    MT_FRJOB,1  Llamada QDOS
            ENDM
*
* Programa pincipal
*
            QDOS    MT_INF,1    Obtengo información
```

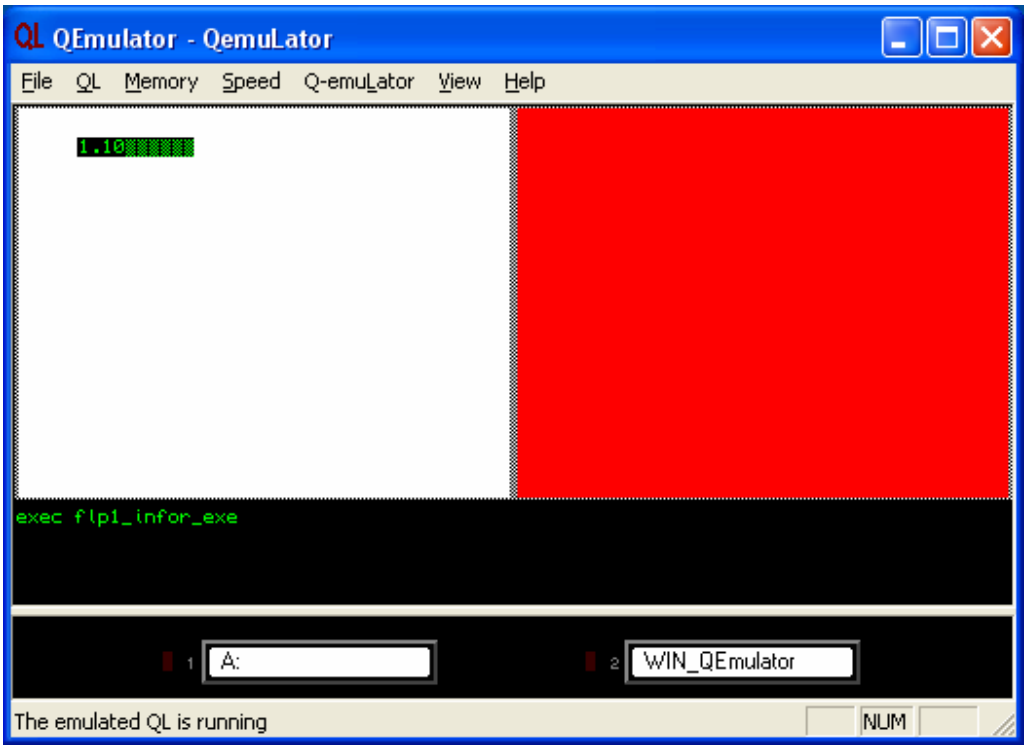
```
            LEA.L   MENSAJE,A3  Apunto A3 a la zona donde guardo infor.
            MOVE.L  D2,(A3)     Llevo A3 el num. de versión
            MOVEQ   #-1,D1      Job actual
            MOVEQ   #2,D3       Nuevo fichero
            LEA.L   DISPOSITIVO,A0  Tipo de dispositivo asociado
            QDOS    IO_OPEN,2   Llamada al QDOS
            MOVEQ   #10,D2      Mando octetos de más
            MOVEQ   #-1,D3      Tiempo de espera infinito
            LEA.L   MENSAJE,A1  apunto a la base de la memoria
            QDOS    IO_SSTRG,3  Llamada al QDOS
*
            FINAL
*
* Sección de datos
*
DISPOSITIVO     DC.W 4
                DC.B 'SCR_'     Dispositivo de salida en la pantalla
MENSAJE         DS.B 10         Base de la memoria

                END
```

Más comentarios:

Vemos que prácticamente se emplean el mismo juego de instrucciones que en el ejemplo anterior, la novedad más importante tal como nos comenta su autor, es el uso de macros. Las macros no forman parte del juego de instrucciones del 68000 sino que son construcciones del ensamblador que nos ahorran el empleo de código repetitivo (sustituyéndose por la macro). Estas macros aceptan una especie de parámetros para flexibilizar y generalizar su uso. Fijémonos también en la sección de datos, una zona donde se reserva memoria DS.B, no para almacenar constantes (DC.z) sino datos generados por nuestro programa, en este caso una cadena de bytes indicativos de la versión del QDOS.

A continuación vemos el resultado de ejecutar el programa en nuestro emulador. Vemos que estamos usando una ROM con la versión 1.10 del sistema operativo.



Paso 4, entrada / salida por consola

Sigamos avanzando en nuestro recorrido y demos el siguiente “pasito”… Ahora le toca el turno a un pequeño programa que hace algo más, pide por consola la entrada de una cadena y luego imprime dicha cadena en la misma consola. El programa lo he tomado del manual de Metacomco y lo podemos ver en el “Listado 3”.

Listado 3, rwstr_asm.

```
*
* Códigos de llamadas al QDOS
*
MT_FRJOB        EQU         $05
IO_OPEN         EQU         $01
IO_CLOSE        EQU         $02
IO_FLINE        EQU         $02
IO_SSTRG        EQU         $07
*
* Macros
*
QDOS            MACRO
                MOVEQ       #\1,D0
                TRAP        #\2
```

```
                ENDM
*
TIDYUP          MACRO
                QDOS        IO_CLOSE,2      Cerrar el canal
                MOVEQ       #-1,D1
                MOVEQ       #0,D3           Cancelar este job
                QDOS        MT_FRJOB,1
*
* En caso de CALL desde SuperBASIC
                MOVEQ       #0,D0           Código de retorno - todo OK
                RTS
                ENDM
*
* Programa principal
*
                MOVEQ       #-1,D1          Job actual
                MOVEQ       #2,D3           Dispositivo exclusivo
                LEA.L       DEVNAME,A0      Puntero a nombre de dispositivo
                QDOS        IO_OPEN,2       Abrir stream
*
* Imprimir etiqueta
*
                MOVEQ       #18,D2          Longitud del string
                MOVEQ       #-1,D3          Timeout infinito
                LEA.L       PROMPT,A1       Puntero a string
                QDOS        IO_SSTRG,3      Imprimir la etiqueta

*
* Leer entrada
*
                LEA.L       BUFFER,A3       Puntero al buffer de entrada
                LEA.L       2(A3),A1        Saltar la primera palabra
                MOVEQ       #30,D2          Longitud del buffer
                QDOS        IO_FLINE,3      Leer input, d1 obt. / estab.
*                                           nbytes leidos
                MOVE.W      D1,(A3)         Grabar los n bytes de la lectura
*
* Print message
*
                MOVEQ       #6,D2           Longitud del texto
                MOVEQ       #-1,D3          Timeout infinito
                LEA.L       MESS,A1         Puntero al mensaje
                QDOS        IO_SSTRG,3      Imprimir mensaje
                MOVEQ       #0,D2           Borrar D2
                MOVE.W      (A3),D2         Longitud del nombre
                LEA.L       2(A3),A1        Puntero al nombre
                QDOS        IO_SSTRG,3      Imprimir nombre
*
* Hacer limpieza
*
                TIDYUP
*
* Sección de datos
*
BUFFER          DS.W            1
                DS.B            30
```

```
DEVNAME         DC.W            4
                DC.B            'CON_'
MESS            DC.B            'Hello '
PROMPT          DC.B            'Enter your name : '
*
                END
```

Comentarios:

Como veis, prácticamente sigue el mismo esquema que el ejemplo anterior, hace el uso de macros para llamadas estándar al QDOS y una macro para la limpieza y finalización del programa. El juego de instrucciones del 68000 empleadas en este ejemplo son las mismas que en los ejemplos anteriores.

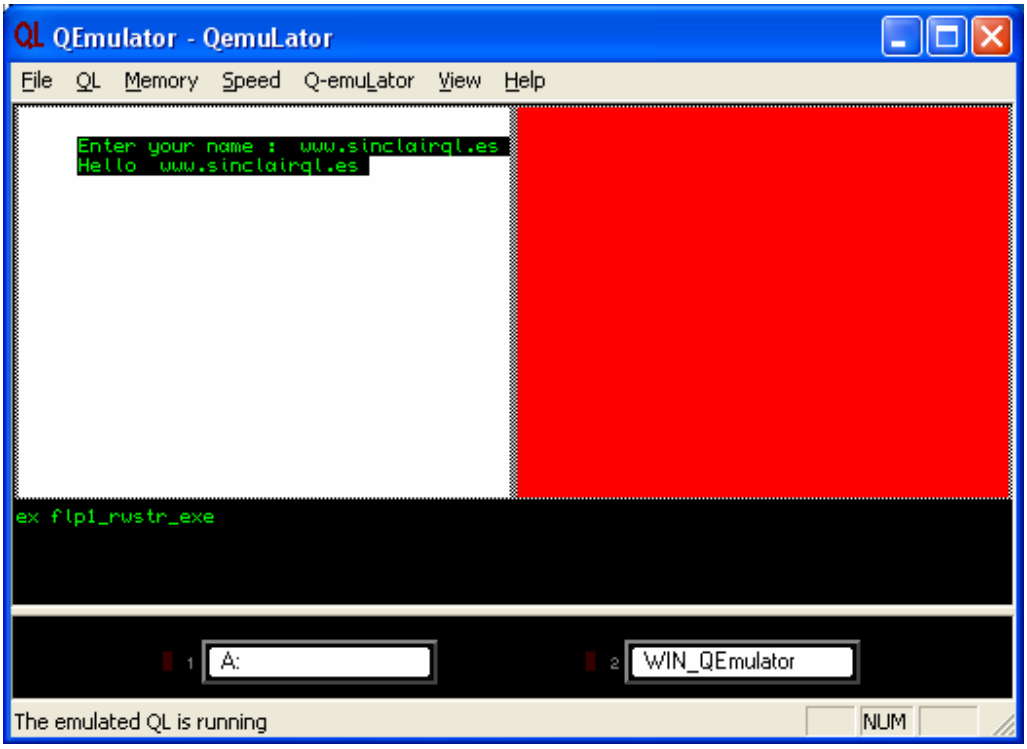
Para la salida de la cadena por pantalla usa:

- TRAP 3 con IO_SSTRG ($07) en D0, la longitud de la cadena en D2 y el puntero del string a imprimir en A1.

Para la entrada por teclado de nuestro "input" usa:

- TRAP 3 con IO_FLINE ($02) en D0, la longitud del buffer en D" y el puntero al buffer de entrada en A3. Este TRAP nos deposita en la dirección de memoria apuntada por A3 el número de bytes leídos y a continuación los bytes de la cadena que hemos tecleado.

La siguiente figura nos muestra la ejecución de este ejemplo.



Paso 5, un volcado de la memoria

Ya hemos visto en los anteriores ejemplos como mostrar cadenas de texto, cómo llamar al QDOS y cómo realizar la entrada salida por consola. En el siguiente programa, también extraído del manual de Metacomo veremos cómo realizar otra función típica en todo programa, esto es la iteración de un bucle un número determinado de veces.

El programa consiste en mostrar por pantalla el contenido de una zona de la memoria de nuestro sistema. Comencemos mostrando el listado del programa (listado 4)

Listado 4, impmem_asm.

```
*
* Códigos de llamadas al QDOS
*
MT_FRJOB        EQU         $05
IO_OPEN         EQU         $01
IO_CLOSE        EQU         $02
IO_SBYTE        EQU         $05
IO_SSTRG        EQU         $07
NADDR           EQU         100
*
* Macros
*
QDOS            MACRO
                MOVEQ       #\1,D0
                TRAP        #\2
                ENDM
*
TIDYUP          MACRO
                QDOS        IO_CLOSE,2      Cerrar canal
                MOVEQ       #-1,D1
                MOVEQ       #0,D3           Cancelar este job
                QDOS        MT_FRJOB,1
*
* En caso de  CALL desde SuperBASIC
                MOVEQ       #0,D0           C–digo de retorno - todo OK
                RTS
                ENDM
*
* Programa principal
*
* Abrir stream
                MOVEQ       #-1,D1          Job actual
                MOVEQ       #2,D3           Dispositivo exclusivo
                LEA.L       DEVNAME,A0      Puntero a nombre de dispositivo
                QDOS        IO_OPEN,2       Abrir stream
*
                LEA.L       $28000,A4       Dirección de comienzo
                MOVEQ       #(NADDR-1),D4   (no. de direcciones a ser
```

```
*                                          examinadas -1)
*
* Bucle principal
*
LOOP
*
* Imprimir dirección con 8 dígitos en hexadecimal
                MOVEQ       #10,D2          Longitud del string
                MOVEQ       #-1,D3          Tiempo de espera infinito
                LEA.L       MESS1,A1        Puntero al stream
                QDOS        IO_SSTRG,3      Imprimir string
*
                MOVE.L      A4,D1           Dirección en D1
                BSR         WRITE32BITS     Imprimir dirección
*
* Imprimir contenido de esa dirección de memoria
                MOVEQ       #14,D2          Longitud del texto
                MOVEQ       #-1,D3          Timeout infinito
                LEA.L       MESS2,A1        Puntero al string
                QDOS        IO_SSTRG,3      Imprimir string
*
                MOVE.L      (A4)+,D1        Puntero a D1 e inc. puntero
*
                BSR         WRITE32BITS     Imprimir contenido
*
* Salida L/F
                MOVEQ       #$0A,D1         L/F en D1
                BSR         WRITECHAR       Escribir el carŒcter
*
* ´Se han examinado todas las direcciones? si no, ir a LOOP
                DBRA        D4,LOOP
*
* Hacer limpieza
*
                TIDYUP
*
*
WRITE32BITS
                SWAP        D1
                BSR.S       WRITE16BITS     Escribir los 16 bit sup. y bajar
                SWAP        D1              a los 16 bits inferiores
*
WRITE16BITS     ROR.W       #8,D1
                BSR.S       WRITE8BITS      Escribir 8 bits (de 16) y bajar
                ROL.W       #8,D1           hasta escribir los 8 bits más
*                                           bajos
*
WRITE8BITS      ROR.B       #4,D1
                BSR.S       WRITE4BITS      Escribir 4 bit sup (de 8) y bajar
                ROL.B       #4,D1           hasta escribir 4 bits inferiores
*
*
WRITE4BITS      MOVE.L      D1,-(SP)        Grabar D1 en la pila
                ANDI.B      #$0F,D1         Máscara para ret. 4 bits inf.
                ADDI.B      #'0',D1         Añadir carácter cero
                CMPI.B      #'9',D1         Si > qué carácter 9
```

```
                BLS.S       WRITE4BITS1     No, entonces escribir carácter
                ADDI.B      #'A'-'9'-1,D1   Si, convertir a rango A-F
WRITE4BITS1
                BSR.S       WRITECHAR
                MOVE.L      (SP)+,D1        Restablecer D1
                RTS
*
*
WRITECHAR
                MOVEM.L     D0/D3/A1,-(SP) Grabar registros en la pila
                MOVE.L      #-1,D3          Timeout infinito
                MOVEQ       #IO_SBYTE,D0    Code a enviar 1 byte
                TRAP        #3              Byte a ser enviado en D1
                MOVEM.L     (SP)+,D0/D3/A1 Restablecer registros
                RTS
*
*
* Sección de datos
*
DEVNAME         DC.W            4
                DC.B            'CON_'
MESS1           DC.B            'Address : '
MESS2           DC.B            '   Contents : '
*
                END
```

Comentarios:

Este nuevo programa emplea básicamente todo lo que hemos estudiado hasta estos momentos pero introduce dos construcciones importantísimas para el desarrollo de programas en ensamblador.

La primera de las construcciones es el equivalente a los bucles FOR (aunque se pueden hacer todo tipo de bucles con esta construcciones WHILE, …). Estos bucles tienen el siguiente esqueleto:

```
            MOVEQ       #N,D4       “Contador” de iteraciones
ETIQUETA

            . . . Sección1
            . . .

            DBRA        D4,ETIQUETA
```

La instrucción fundamental en este recorte de código es la sentencia DBRA que en la mayoría de los ensambladores equivale también a DBF. Esta instrucción y otras pertenecen a la familia de instrucciones tipo DBcc donde cc hace referencia al código de condición del registro de estado del 68000. Estos códigos de condición pueden ser por ejemplo CC (acarreo a cero), EQ (igual a), LE (menor que), RA- (falso), …

En nuestro recorte de código anterior, la sentencia `DBRA D4,ETIQUETA` provoca que la ejecución del programa salte a ETIQUETA hasta que D4 sea igual a -1, si no es así entonces D4 se decrementa (D4 -1). En nuestro ejemplo el trozo de código *"Sección1"* (localizado entre ETIQUETA y DBRA) se ejecutará N veces.

La segunda construcción de interés en nuestro ejemplo es la construcción equivalente a llamar a una subrutina (algo así como el DEFine PROCedure de SuperBASIC). Esta construcción tiene el siguiente esqueleto.

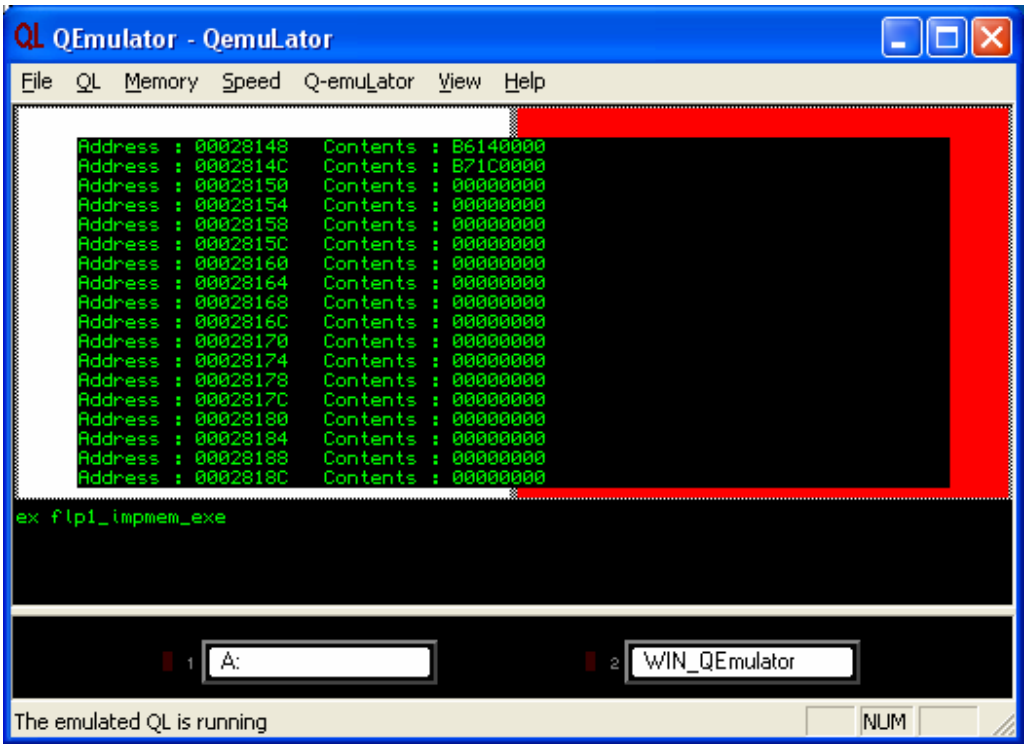
```
            BSR         ETIQUETA
            . . . sección_1
            . . .

ETIQUETA
            . . . sección_2
            . . .
            RTS
```

En nuestro recorte de código, el programa cuando llega a la instrucción BSR (Branch to SubRoutine) deriva el hilo de la ejecución a la zona de código marcada como ETIQUETA y ejecutará la seccion_2. Una vez que la ejecución llegue a RTS (ReTurn from Subroutine) el hilo de ejecución retornará a la línea siguiente a la instrucción que hizo la llamada y continuará en sección_1. Es responsabilidad del programador conservar el estado de los registros en la llamada y recuperarlos cuando se retorna (para ello nos ayudaremos de la pila del usuario).

Existen variantes de la instrucción RTS que restauran de forma automática el registro de estado, también el 68000 nos ofrece facilidades para almacenar en una sola instrucción los registros deseados en la pila, así como facilidades para recuperar dichos datos de la pila posteriormente.

La siguiente imagen muestra el programa en acción.



Paso 6, control del teclado

Le llega el turno al último programa de ejemplo en este recorrido por el lenguaje ensamblador para el 68000. Es un ejemplo extraído del manual del Assembler Development Kit de Metacomco (como no). El programa nos muestra una caja en la pantalla en cuyo interior se muestra una cadena de texto que va haciendo scroll horizontal cuando dicho cuadro no está en movimiento. Con las teclas cursor arriba, cursor abajo, cursos derecha y cursor izquierda podemos desplazar la caja sobre la pantalla. Empleando las teclas ALT-cursor arriba, ALT-cursor abajo, ALT-cursos derecha y ALT-cursor izquierda podemos alterar el tamaño de la caja en altura y anchura. También se lleva un control sobre los límites del área de dibujo para que los desplazamientos no se salgan del área preestablecida para el movimiento de la caja.

Veamos el listado de este programa.

Listado 5, cajascr_asm.

```
*
* Sección de constantes
*
*
                TTL             Demostración - constantes
```

```
*
* Llamadas QDOS
*
* Códigos Trap #1
*
MT_FRJOB        EQU     5
*
* Códigos Trap #2
*
IO_OPEN         EQU     1
IO_CLOSE        EQU     2
*
* Códigos Trap #3
*
IO_FBYTE        EQU     1
IO_SBYTE        EQU     5
SD_BORDR        EQU     $C
SD_WDEF         EQU     $D
SD_CURE         EQU     $E
SD_PIXP         EQU     $17
SD_PAN          EQU     $1B
SD_CLEAR        EQU     $20
SD_SETIN        EQU     $29
SD_SETSZ        EQU     $2D
*
* Colores
*
C_GREEN         EQU     4
C_WHITE         EQU     7
C_CHAR1         EQU     C_GREEN
C_CHAR2         EQU     C_WHITE
*
* Códigos de teclas
*
K_ENTER         EQU     $0A
K_SPACE         EQU     $20
K_UP                EQU     $D0
K_DOWN          EQU     $D8
K_LEFT          EQU     $C0
K_RIGHT         EQU     $C8
K_ALT_UP        EQU     $D1
K_ALT_DOWN      EQU     $D9
K_ALT_LEFT      EQU     $C1
K_ALT_RIGHT     EQU     $C9
*
* Algunas constantes útiles
*
CHWIDTH         EQU     16
CHHEIGHT        EQU     20
MINH            EQU     CHHEIGHT+2
MINW            EQU     CHWIDTH*6+4
MAXX            EQU     512-CHWIDTH-4
MAXY            EQU     256-CHHEIGHT-2
BWIDTH          EQU     1           Ancho del borde de la ventana
WIDTH           EQU     MINW
HEIGHT          EQU     MINH
```

```
X               EQU     304
Y               EQU     200
*
* Nombres para registro
*
WW          EQUR        D4          Ancho y alto de la ventana
WH          EQUR        D5          actual
WX          EQUR        D6          Posición X,Y de la esquina
WY          EQUR        D7          superior izquierda
            PAGE
            TTL     Demostración - Macros
            SPC     2
*
* Macros
*
            SPC     2
*
* Una llamada genƒrica para el posicionamiento del cursor, scrolling
* panning, etc. El timeout es infinito y se asume que A1 contiene
* el canal a la pantalla
*
SCROPS      MACRO
            MOVEQ       #\1,D1
            MOVEQ       #\2,D2
            MOVEQ       #\3,D0      Opcode en D0
            MOVEQ       #-1,D3      Timeout
            LEA.L       WINDOW,A1   Dirección del bloque de ventana
            TRAP        #3          Hacerlo
            ENDM
            SPC         2
*
* Una llamada generalizada para peticiones al QDOS
*
QDOS        MACRO
            MOVEQ       #\1,D0      Código del trap
            TRAP        #\2
            ENDM
*
TIDYUP      MACRO
            QDOS        IO_CLOSE,2  Cerrar el canal de la consola
            MOVEQ       #-1,D1
            MOVEQ       #0,D3       Cancelar este job
            QDOS        MT_FRJOB,1  Terminar este job
*
* En caso de uso de CALL
            MOVEQ       #0,D0
            RTS
            ENDM
*
            PAGE
            TTL         Demotración - Programa principal
*
* Programa principal
*
INIT
*
```

```
* Abrir el stream de consola
*
            MOVEQ       #-1,D1      Job actual
            MOVEQ       #2,D3       Dispositivo exclusivo
            LEA.L       DEVNAME,A0  Puntero al nombre del dispositivo
            QDOS        IO_OPEN,2   Abrir canal, ID en A0
*
* Establecer la ventana por defecto

            SCROPS      3,1,SD_SETSZ    Caracteres grandes
            SCROPS      0,0,SD_CURE     Activar cursor
*
* Inicializar la sección de ventana
*
            MOVE.W      (A1)+,WW        Ancho
            MOVE.W      (A1)+,WH        Alto
            MOVE.W      (A1)+,WX        X
            MOVE.W      (A1)+,WY        Y
            LEA.L       LOGO,A4         Uar A4 para apuntar al carŒcter acutal
            LEA.L       LOGOE,A5        y A5 para mantener el final
            BRA         SETWIN9         Mostrar ventana y entrar en el bucle
*
* Bucle principal
*
SETWINDOW
            BSR         VDU_RDCH        Obtener carácter o espera
            BEQ.S       SETWIN0         Carácter leído, tratarlo
*
* No se ha tecleado carácter todavía, manejar el scroll del texto
*
            SCROPS      -CHWIDTH,0,SD_PAN
            MOVE.B      10(A4),D1       Obtener color para este carácter
            QDOS        SD_SETIN,3      Cambiar color
            MOVE.B      (A4)+,D1        Obtener carácter
            QDOS        IO_SBYTE,3      Imprimir carácter (A0 tiene Id canal)
            BSR         SETCURSOR       Cursor atrás
            CMPA.L      A4,A5           Se ha alcanzado el final?
            BNE.S       SETWINDOW       No, fantástico
            LEA.L       LOGO,A4
*
* Manejar el movimiento de la ventana
*
SETWIN0
            CMPI.B      #K_ENTER,D1     Tecla ENTER?
            BEQ         FINISH          Fin de programa
            CMPI.B      #K_SPACE,D1     Tecla ESPACIO?
            BEQ         SETWIN9         Si, redibujar la ventana
SETWIN1
            CMPI.B      #K_UP,D1        Tecla arriba?
            BNE.S       SETWIN2
            CMPI.W      #CHHEIGHT,WY    Comprobar Y <= CHHEIGT
            BLT.S       SETWINDOW       Si es as“ no alterar, esperar sig. car.
            SUBI.W      #CHHEIGHT,WY    En caso contrario, tomar CHHEIGHT
            BRA         SETWIN9         Redibujar ventana
SETWIN2
            CMPI.B      #K_DOWN,D1      Tecla abajo?
```

```
            BNE.S       SETWIN3
            MOVE.W      WH,D3           D3 = Alto
            ADD.W       WY,D3           Añadirle a la Y
            CMPI.W      #MAXY,D3        Es Y + Alto >= MAXY ?
            BGE         SETWINDOW       Si, espera por siguiente carácter
            ADDI.W      #CHHEIGHT,WY    No, añadir CHHEIGHT a Y
            BRA         SETWIN9         Redibujar ventana
SETWIN3
            CMPI.B      #K_LEFT,D1      Tecla izquierda?
            BNE.S       SETWIN4
            CMPI.L      #CHWIDTH,WX     Es X < ancho ?
            BLT         SETWINDOW       Si, entonces ignorar
            SUBI.W      #CHWIDTH,WX     X=X-ancho
            BRA.S       SETWIN9         Redibujar ventana
SETWIN4
            CMPI.B      #K_RIGHT,D1     Tecla derecha?
            BNE.S       SETWIN5
            MOVE.W      WW,D3           Ancho
            ADD.W       WX,D3           sumar X
            CMPI.W      #MAXX,D3        Es el final de la pantalla?
            BGE         SETWINDOW       Si, ignorar
            ADD.W       #CHWIDTH,WX     X=X+ancho
            BRA.S       SETWIN9         Redibujar ventana
SETWIN5
            CMPI.B      #K_ALT_UP,D1    ALT+tecla arriba?
            BNE.S       SETWIN6
            CMPI.W      #MINH,WH        Es alto<=MINH?
            BLE         SETWINDOW       Si, ignorar
            SUB.W       #CHHEIGHT,WH    Alto=Alto-CHHEIGHT
            BRA.S       SETWIN9         Redibujar la ventana
SETWIN6
            CMPI.B      #K_ALT_DOWN,D1   ALT+tecla abajo?
            BNE.S       SETWIN7
            MOVE.W      WH,D3           Alto
            ADD.W       WY,D3           Añadir Y
            CMPI.W      #MAXY,D3        Es Alto + Y >= MAXY ?
            BGE         SETWINDOW       Si, ignorar
            ADD.W       #CHHEIGHT,WH    Alto = alto = CHHEIGHT
            BRA.S       SETWIN9
SETWIN7
            CMPI.B      #K_ALT_LEFT,D1   ALT+tecla izquierda ?
            BNE.S       SETWIN8
            MOVE.W      WW,D3           Ancho
            SUB.W       #CHWIDTH,D3     Menos CHWIDTH
            CMPI.W      #MINW,D3        Es <= MINW ?
            BLT         SETWINDOW       Si, ignorar
            SUB.W       #CHWIDTH,WW     Reducir ancho
            BRA.S       SETWIN9         Redibujar ventana
SETWIN8
            CMPI.B      #K_ALT_RIGHT,D1  ALT+tecla derecha ?
            BNE         SETWINDOW
            MOVE.W      WW,D3           Ancho
            ADD.W       WX,D3           Sumar X
            CMPI.W      #MAXX,D3        Es >= MAXX
            BGE         SETWINDOW       Si, ignorar
            ADD.W       #CHWIDTH,WW     Incrementar ancho
```

```
SETWIN9
            SCROPS      0,1,SD_BORDR    Elimiar borde antiguo
            SCROPS      0,0,SD_CLEAR    Y borrar ventana antigua
            LEA.L       WINDOW,A1       Obtener la dirección del buffer
            MOVE.W      WW,(A1)+        Restablecer ancho
            MOVE.W      WH,(A1)+        Restablecer alto
            MOVE.W      WX,(A1)+        Restablecer X
            MOVE.W      WY,(A1)         Restablecer Y
            SCROPS      C_WHITE,BWIDTH,SD_WDEF    Redefinir ventana
            SCROPS      0,0,SD_CLEAR    Borrar
            BSR.S       SETCURSOR       Cursor a la posición correcta
            LEA.L       LOGO,A4         Reset puntero a carácter
            BRA         SETWINDOW       Regresar y obtener siguiente tecla
*
* Limpieza
*
FINISH
            TIDYUP
*
* Posición del cursos al lugar estándar (PIXP especifica ventana relativa)
*
SETCURSOR
            MOVEQ       #0,D2           Cursor en posición superior
            MOVE.W      WW,D1           Borde derecho coordenadas-x
            SUBI.W      #CHWIDTH*2+5,D1 Dos car. separado del borde
            MOVEQ       #-1,D3          Timeout
            QDOS        SD_PIXP,3       Posición del cursor
            RTS
*
* Leer un carácter del teclado
*
VDU_RDCH
            MOVEQ       #30,D3          Algún tiempo de espera
            QDOS        IO_FBYTE,3      Obteher carácter en D1.B (A0 id canal)
            TST.L       D0              Comprobar timeout=0 si car. tecleado
            RTS
*
* Area de datos
*
* Window co-ordinates
*
WINDOW      DC.W            WIDTH
            DC.W            HEIGHT
            DC.W            X
            DC.W            Y
*
DEVNAME     DC.W            22
            DC.B            'CON_100x22a304x200_128'
LOGO        DC.B            'MetaComCo'
LOGOE       DC.B            C_CHAR1,C_CHAR2,C_CHAR2,C_CHAR2
            DC.B            C_CHAR1,C_CHAR2,C_CHAR2
            DC.B            C_CHAR1,C_CHAR2,C_CHAR2
*
            END
```

Comentarios:

Este programa es bastante más largo pero más completo. Nos muestra cómo controlar las teclas del cursor del teclado. Hace uso extensivo de subrutinas y macros ya comentados en ejemplos anteriores y además incluye algunos nemotécnicos nuevos con el fin de sumar y restar valores. Entre estos nemotécnicos nuevos están:

ADD y sus variantes ADDI, empleadas para operaciones de suma.
SUB y sus variantes SBBI, empleadas para operaciones de resta
CMP y CMPI, empleadas para ejecutar comparaciones entre registros y/o posiciones de memoria.

Otro aspecto a destacar en éste último ejemplo es el empleo de ciertas directivas del compilador, como por ejemplo PAGE, TTL, SPC, … Estas directivas no forman parte del juego de instrucciones del 68000 sino que son propias del ensamblador. Sirve para dirigir las acciones del propio ensamblador y muchas de ellas son de uso común.

A continuación una imagen del programa en acción.

Para terminar se detallan una lista casi completa de las directivas del ensamblador a modo de resumen (no forman parte del juego de instrucciones del 68000). Se muestran con ejemplos, algunas ya han sido comentadas, otras son nuevas. Esta relación ha sido extraída de la revista Qlave en uno

de sus primeros números donde se hacía un análisis en profundidad del ensamblador de Metacomco.

ORG #30000 Indica origen absoluto del programa o los datos en #30000. Será preciso cargar lo etiquetado así con LBYTES.

RORG O Indica origen relativo del programa, de forma que el contador de programa es colocado a ese valor y las instrucciones que le siguen se sitúan de forma relativa a éste. Si no indicamos nada es como si hubiésemos colocado este RORG 0 con lo que el PC tiene el valor 0 inicialmente. Los programas que contienen éste son ejecutables vía EXEC. Habitualmente no se suele colocar ORG ni RORG con lo que el programa es ejecutable tanto vía EXEC como por CALL (previo LBYTES).

SIZE 1000 Indica el espacio de datos a reservar como interno al programa, recordar que todo programa ejecutable por EXEC debe contener espacio necesario para sus datos y el stack. Si no se incluye se supone un espacio de 500 octetos.

END Se coloca para indicar el fin del programa, si no se coloca genera un mensaje tipo Warning al finalizar el compilado.

EQU Asigna el valor de la expresión del campo a la etiqueta colocada en la instrucción. La asignación es permanente y la expresión no debe contener referencia a otras zonas del programa (ejemplo CASA EQU 12000, CASA tiene el valor 12000 en toda la ejecución del programa, dicho valor no se puede cambiar).

EQUR Permite asignar a un nombre creado por nosotros uno de los registros del procesador, no es posible renombrar SR, CCR, USP. La asignación es permanente y no es redefinible en ninguna otra zona del programa. En el ejemplo se asigna el nombre DATO a D0, Ej: DATO EQU D0.

SET Permite asignar a una etiqueta una expresión de igual forma que lo hace EQU con la diferencia de que el valor puede ser alterado a lo largo del programa. La expresión no deberá contener referencia a otras zonas del programa. Ej: CASA SET 12000, asigna 12000 a CASA.

DC Defina valores constantes en un programa, pueden tener un número indefinido de operandos separados por comas. Son semejantes a los DATAS de un programa BASIC Es preciso asignarles la especificación de Byte, Palabra o Palabra Larga. Se les puede colocar una etiqueta de forma que una referencia a este espacio de datos desde el programa se hace por medio de la etiqueta.

```
CASA  DC.B   6,7,8,9,4
MESA  DC.W   240, 300
ASA   DC.L   70000
```

De esta forma si sitúo el registro A0 en CASA, con LEA CASA,A0 la celdilla de memoria a la que apunta A0 contendrá 6, la celdilla a la que apunta A0+1 contendrá 7, A0+2 8 ... De igual modo si hago LEA MESA,A0 , la celdilla a la que apunta A0 contendrá 240, A0+2 contendrá 300.

DS — Permiten reservar zonas de memoria vacías que posteriormente serán rellenadas con datos. Igual que las anteriores las referencias a las mismas se realizarán vía la etiqueta.

```
CASA   DS.B   5   Reserva 5 octetos de memoria
MESA   DS.W   4   Reserva 4 palabras (8 octetos)
ASA    DS.L   2   Reserva 2 palabras largas (8 octetos)
```

PAGE — Avanza el listado en ensamblador hasta la siguiente página. Este comando no aparece en los listados generados.

LIST — Hace que a partir de donde esté los comandos sean listados. Por defecto se genera siempre listado.

NOLIST — Desactiva el listado de forma que los comandos que le siguen no son listados hasta que aparezca LIST.

SPC — Hace que aparezcan las líneas en blanco que se deseen en un listado. Ej: SPC 10 , si deseamos 10 líneas en blanco.

NOPAGE — Desactiva el avance de página y la colocación de cabecera en los listados.

LLEN — Coloca la longitud de na liena en listado ensamblador, la longitud debe estar entre 60 y 132. El defecto es 132. Ej: LLEN 90.

PLEN — Coloca la longitud de la página de listado. El valor debe estar entre 24 y 100. El defecto es 60.

TTL — Asigna una cadena como título de un programa. Será reproducida cada avance de página. La cadena no deberá tener más de 40 caracteres. Ej: TTL Programa ensamblador.

NOOBJ — Desactiva la generación de código objeto hasta el fin del programa aun cuando se haya indicado un fichero en el inicio de la ejecución del compilador.

FAIL — Genera un error de usuario.

CNOP — Permite que una sección de código sea alineada a un límite determinado. En particular permitirá que una estructura de datos se alinee a un límite de palabra larga.

```
CNOP  0,4  alinea el código al límite de la próxima palabra larga.
```

CNOP 2,4 alinea el código a un límite de palabra de 2 bytes antes de la más cercana alineación de palabra larga.

IFEQ
IFNE — Permite activar o desactivar el ensamblado dependiendo del valor de la expresión. Así si el operando es cero (EQUAL), IFEQ activa el ensamblado si no lo fuera lo desactivaría. IFNE se comporta de modo contrario.

ENDC — Marca el fin del un ensamblado condicional de las características mencionadas arriba.

MACRO — Marca el comienzo de una macro-definición, ejemplos de ellas se pueden ver en los ejemplos de este artículo. Contienen un conjunto de instrucciones que se ejecutan cada vez que se la llama tecleando la etiqueta que se coloca a su comienzo. Son semejantes a las funciones y procedimientos del SuperBasic. De igual forma admiten parámetros. Para indicar un parámetro se coloca un símbolo #seguido por el número del parámetro que llevemos usados; 1 o 2 o ... Dentro de una macro definición no se puede comenzar la definición de otra macro, es preciso definirlas separadamente y es posible que una macro use otras macros, ahora bien hay un límite en el nivel de anidación que se puede usar, este nivel es de diez.

ENDM — Indica el fin de la macro-definición.

MEXIT — Permite forzar la salida de una macro durante su ejecucion. Se suele usar conjuntamente con IFEQ y IFNE.

XDEF — Nos ayuda a crear ficheros que pueden ser lincados entre si por el lincador incluido en el paquete. Así podemos encontrarnos un fichero:
XDEF DECIMAL,SIST,EV
Esto nos indica que este fichero contiene las definiciones de tres subrutinas que podrán ser usadas por otros programas. DECIMAL, SIST, EV son las etiquetas que habremos colocado al comiento de la subrutinas.

XREF — Es el comando que debemos incluir en el caso de que en un fichero vayamos a usar subrutinas que no estén escritas en el programa y que se incluirán más tarde al enlazar éste fichero con el que contenga esas rutinas.

GET — Permite insertar ficheros externos en el fichero fuente actual. Ver que esto es completamente distinto a las referencias anteriores ya que aquí se inserta código fuente mientras que antes enlazábamos código objeto. Este comando es útil para insertar ficheros.