Tradúzcamelo

Niveles de compromiso
Desde el punto de vista del programador, se puede considerar el ordenador como tres "máquinas virtuales". A un nivel 3, el ordenador del ejemplo funciona en PASCAL. A nivel 2, funciona en assembler, y a nivel 1 el código máquina directamente se enfrenta con el hardware

Este capítulo final sobre programación del 68000 está dedicado al estudio de la operativa del ensamblador

A lo largo de esta serie hemos empleado con frecuencia las instrucciones a nivel ensamblador para ilustrar las características del microprocesador 68000. En nuestros análisis supusimos que el ordenador obedece estas instrucciones según van apareciendo en su forma ensamblador. La máquina interpreta de hecho un código binario que representa las instrucciones ensamblador de nivel más alto, siendo tarea del ensamblador el *traducir* estas instrucciones a código máquina.

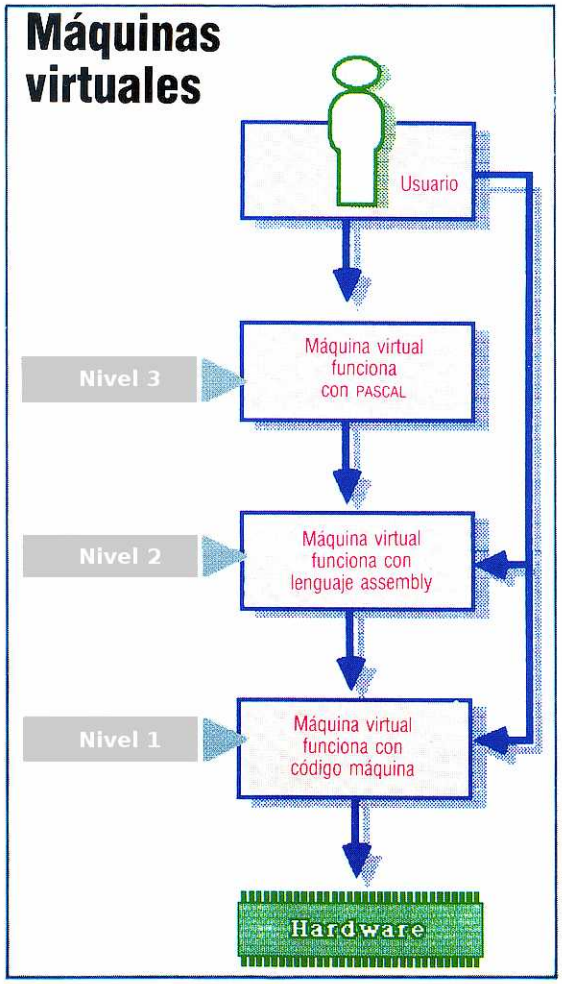
Sin embargo, es conveniente considerar la ejecución de nuestros programas como si la máquina los ejecutara a nivel fuente. Este nivel puede ser muy alto (p. ej., los paquetes de aplicación como el *WordStar*) o ligeramente inferior (como los programas en PASCAL) o muy bajo, casi equiparable al nivel de los programas de ensamblador. Esto explica el enfoque a varios niveles de nuestra visión de la máquina, que se compone de estratos discretos o niveles que se convierten en máquinas *virtuales*, ejecutoras de un determinado lenguaje de programación, como se ilustra en el diagrama (derecha).

El modo como un programa funciona a cualquier nivel es uno de éstos:

- *Traducción*: Un programa que está a un nivel se traduce al nivel inferior de la máquina. Por ejemplo, podemos traducir un programa PASCAL de nivel 3 en ensamblador para ser tratado a nivel 2 por un compilador.
- *Interpretación*: Un programa en un nivel es interpretado por un programa (denominado *intérprete*, claro está) que funciona en una máquina de nivel inferior. Por ejemplo, la máquina interpreta los modelos binarios de bits que representan las instrucciones para operaciones de máquina (o nivel de registro). Igualmente, el nivel 2 puede interpretar también programas en PASCAL desde el nivel 3.

Lo que más nos interesa aquí es el proceso de traducción. Hay muchas formas de traducción que van desde los compiladores hasta los ensambladores, pero todas tienen una cosa en común: todas toman instrucciones de alto nivel y proporcionan las correspondientes instrucciones a nivel inferior. Las instrucciones fuente originales no se precisan entonces, y en cierto sentido son "tiradas a la papelera" por el traductor (sin embargo, un intérprete todavía tendría necesidad del programa fuente original).

El objetivo del ensamblador es traducir las instrucciones de ensamblador en alto nivel a código binario ejecutable por la máquina. Por ejemplo:




```
MOVE.W D3,D5
```

se traduciría así:

```
0011 101 000 000 011
```

El 0011 corresponde a "mover una palabra" (MOVE.W); 101 000 corresponde a "al D5"; y 000 011 corresponde a "desde el D3".

Pero el trabajo a nivel de codificación de bits es extraordinariamente tedioso y propenso a errores, por lo que como mínimo necesitamos expresiones mnemotécnicas que nos permitan recordar las instrucciones y los objetos de datos. Por ejemplo, MOVE.W TOTAL,D4 significa que nos podemos referir a posiciones de memoria mediante nombres simbólicos que nos indiquen el significado del contenido de esa posición (TOTAL en este ejemplo).

Los tipos de error que podríamos cometer si codificáramos los bits a mano serían, entre otros:

- Equivocar los códigos de instrucción (opcodes).
- Direccionamiento absoluto incorrecto.
- Asignar un número inexacto de bytes por instrucción.

Esto quiere decir que no hablaremos siquiera del caso en que haya que hacer una traducción manual que supere media página de instrucciones. De aquí la importancia desempeñada por el ensamblador en la traducción del código fuente al código objeto.

El proceso assembly

El proceso assembly se inicia cuando el ensamblador lee las sentencias fuente en el archivo de texto fuente (texto escrito en ASCII) y produce un archivo de listado (o lo imprime) más un archivo objeto (o lo carga en la máquina). Nuestro archivo Formato Listado Assembly, que ilustra los elementos de una traducción, es un programa que ejecuta un cálculo aritmético en los elementos de una tabla.

Varios son los aspectos dignos de nota en el listado. Primero, usted establece una lista de contenidos de las posiciones y después el resto de la línea continúa con el código fuente original que sigue a un número insertado de sentencia. La información de errores se referirá al número de sentencia. Si la línea 14 contiene un error, se imprimirá E en esa línea y se hará una referencia a esa línea en la sección de información de errores del listado.

El formato del archivo binario es interesante por varias razones. Ante todo, el archivo ha de contener la información binaria en forma codificada, así como información del cargador para poder cargar correctamente el código binario en la memoria. La manera de conseguir esto es almacenando la información de direcciones y contenidos en binario codificado en hexa, por lo que el cargador debe conocer el formato binario. El formato exacto depende del ensamblador, pero la compatibilidad del cargador es esencial.

La última parte del listado es la impresión de la tabla de símbolos. Esta tabla proporciona los valores numéricos asignados a los niveles declarados en el programa. Por ejemplo, INPUT (definido en la sentencia 25) tiene valor 1024 (hexa) y está referenciado en la sentencia 10. Son todos detalles casi triviales en este pequeño programa de ejemplo, pero en un programa algo más extenso se convierten en información esencial para la depuración de errores.

El destino de la salida depende de donde esté funcionando el ensamblador (naturalmente, el ensamblador es un programa como otro cualquiera). Si, por ejemplo, estamos usando un método de *ensamblador cruzado*, el ensamblador funciona en un sistema huésped como el Unix, y el listado y la información binaria estarán en archivos. El archivo binario ha de ser cargado en el 68000 de destino antes de poder ejecutar el programa.

Otro modo de hacer esto es ejecutar el ensamblador en la máquina destino, si ésta tiene al menos algún tipo de sistema de gestión de archivos, aunque sea rudimentario. A veces los códigos binarios se cargan directamente en la memoria con el assembly destino y los listados pueden ser impresos directamente si está activada la impresora.

Los mecanismos del ensamblador

Los mecanismos que emplea un ensamblador nos dan idea de los problemas que pueden presentarse (¡que inevitablemente se presentarán!). Por lo general, el proceso de assembly es una organización de *dos pasos*. En el primer paso, el ensamblador:

- Decodifica los mnemotécnicos assembly (MOVE, ADD, MULS y demás).
- Cuenta los bytes de dirección (para poder calcular las direcciones).
- Construye una tabla de símbolos (que nos dice los valores asignados a cada símbolo).
- Descarta cualquier error de sintaxis (p. ej., ADDQQ es una instrucción ilegal).

Cuando el ensamblador lee la sentencia 9 en el primer paso por el programa *Formato listado assembly*, el contenido de etiqueta de la posición será 41F8. Que corresponde a un "movimiento rápido" a D0 en el modo inmediato con una constante de 12 (desde LENGTH en la tabla de símbolos).

Para la sentencia 10, el ensamblador puede establecer la posición 1002 con 41F8 para la instrucción LEA, pero no puede establecer todavía la dirección de INPUT. Así, la posición 1004 se deja vacía hasta el segundo paso cuando se conoce la dirección de INPUT.

En el segundo paso, el ensamblador puede sustituir las direcciones ahora conocidas dondequiera que se empleen dentro del programa, y dar salida al código binario y listado del programa. Pueden también darse errores que depurar, pero por lo general se atienden en el primer paso.

Formato listado assembly

```
POS     OBJETO                          SENT                    SENTENCIA FUENTE
                                        1    *Esta sentencia toma cada elemento de una tabla
                                        2    *y lo convierte en
                                        3    *                  INPUT[I]:=2*INPUT[I]+3
                                        4
                                        5
001000                                  6          ORG $1000
        =000C                           7    length: equ 12
                                        8
001000  700C                            9    start: moveq  #length,d0
001002  41F8  1024                      10          lea    input,a0
001006  4244                            11          clr    d4
001008  4243                            12          clr    d3
                                        13
00100A  3610                            14   loop:  move.w (a0),d3
00100C  C7FC  0002                      15          muls   #2,d3
001010  D67C  0003                      16          add.w  #3,d3
001014  30C3                            17          move   d3,(a0)+
001016  D843                            18          add    d3,d4
                                        19
001018  5340                            20          subq   #1,d0
00101A  6600  FFEE                      21          bne    loop
00101E  31C4  103C                      22          move   d4,sum
001022  4F40                            23          trap   #0
                                        24
001024  0001  0002  0003  0004          25   input; dc.w   1,2,3,4,5
        0005
00102F  0006  0007  0008  0009          26          dc.w   6,7,8,9,10
        000A
001038  000B  000C                      27          dc.w   11,12
                                        28
00103C  0000                            29   sum:   dc.w   0
                                        30
                                        31          end
No se encontraron errores en este ensamblaje

        SÍMBOLOS                        DEFN        VALOR   REFERENCIAS
INPUT                                   25          1024    10
LENGTH                                  7           000C    9
LOOP                                    14          100A    21
START                                   9U          1000
SUM                                     29          103C    22
```

Podemos emplear asimismo el ensamblador como una útil calculadora, creando las expresiones simbólicas que calculan el valor de un operando. Por ejemplo, estas líneas introductorias de un listado assembly:

```
LENGTH   DC.W     (START-ARRAY)*4
ARRAY    DC.W     1,2,3,4,5
START    MOVE.W   LENGTH,D3
```

Aquí necesitamos calcular la longitud de la tabla en bytes (START-ARRAY multiplicada por 4), para emplearlo en el programa (para poner 3 en el registro de datos en este caso). Si más tarde cambiamos en algún momento la longitud de la tabla dentro del programa en ensamblador, LENGTH se restablecerá automáticamente.

Ayudas de documentación

Otra ayuda que proporciona el ensamblador está en las áreas de documentación. Es posible agregar comentarios a nuestros programas que orienten a los lectores sobre el significado de las instrucciones, y también podemos incluir detalles sobre registros convenidos y diseño del programa, por ejemplo. Véase el siguiente listado:

```
*Programa para entrar una cadena de caracteres
 y verificarla
*El texto es entrado a través de ACIA y verificado
*A. Gómez - Dic 85
*
*Registros convenidos
*
*Registros de direcciones
*       A1 puntero para entrar cadena
*       A2 puntero para almacenar teclado
*       A3 puntero para carácter actual
*Registros de datos
*       D0 entrada carácter
*       D1 salida carácter
*
START JSR  INIT          inicializa la E/S
      JSR  READSTRING    lee cadena entrada
      JSR  VALIDATE      y la comprueba
      JSR  SIGNAL        señal si es correcta
      BRA  START         bucle indefinido
```

Leyendo estos comentarios nos informamos sobre el programa, su estructura, los registros convenidos. Probablemente sería pedir demasiado si se exigieran aún más detalles a este nivel, pero es claro que el lector puede descender a otro nivel y mirar los comentarios contenidos en el código de la subrutina si necesita más información.

Lo importante es que el programa quede razonablemente bien documentado con el empleo de comentarios.

Directivas del ensamblador

También el assembler nos permite otras ayudas a la programación en el área general de las *directivas del ensamblador*. Pueden ser meras ayudas documentales como TTL o "directiva del título", que ordena la impresión de un determinado título en cada página del listado, o un medio para pasar la información al ensamblador. Por ejemplo, puede que deseemos señalar el final de nuestro texto de entrada para que el ensamblador comience su ensamblaje. Nuestro módulo del programa contendrá las siguientes líneas:

```
*Los comentarios de documentación
*Con nuestro nombre y fecha por lo menos
        TTL   Éste es mi programa
        ORG   $1000   *una instrucción de origen

START   MOVE  D1,D2   *y algunas instrucciones
                      *y otras y otras más
        END           *y la directiva de final
```

Las más importantes directivas del ensamblador se resumen en la tabla de más abajo.

Formato del ensamblador

No es de extrañar que los ensambladores construidos por distintos fabricantes difieran en detalles en cuanto a su operativa y formato. No obstante, la mayoría de los ensambladores se ajustan a una estructuración fuente semejante a la que sigue. Primero, los comentarios pueden ponerse empleando el signo * como primer carácter de la línea, o después de una instrucción si entre ésta y el comentario media un espacio como mínimo. Así se permiten estas dos líneas:

```
*Esto sólo es un comentario
START   ADD   D1,D2 *y esto, otro
```

Lo que constituye una instrucción debe naturalmente definirse, y consta de tres campos opcionales:

- *Campo de etiqueta*: Cada nombre empleado como una etiqueta debe comenzar con un carácter alfabético y tendrá de longitud total menos de treinta caracteres alfanuméricos. Se notará que si se omite este campo se ha de insertar al menos un espacio.
- *Campo del opcode*: Se empleará una del conjunto de instrucciones del 68000.
- *Campo del operando*: Los operandos de instrucción legales deben usarse después de un espacio al menos tras el campo del opcode. Pero en el campo del operando no deben existir espacios aun en el caso de emplear dos operandos.

He aquí algunos ejemplos de sentencias no permitidas en ensamblador:

Tabla de directivas del ensamblador

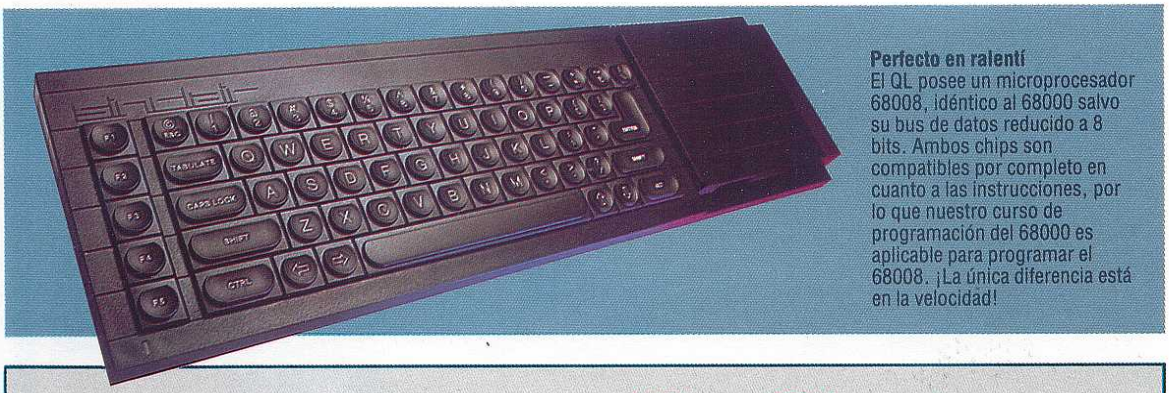
ORG	Abreviatura de "origen". Especifica la dirección de inicio del código
RORG	Vale lo dicho para ORG, pero con código relativo del PC
TTL	Pone un título en cada página del listado
END	Fin del texto para ensamblar
EQU	"Igual a" (*equate*): asigna un valor al símbolo especificado (p. ej., LENGTH EQU 10+50*ARRAY1, donde LENGTH vale 610 suponiendo que ARRAY1 vale 12)
DC	"Declarar Constante": asigna un valor constante (byte, palabra o longitud palabra larga). P. ej., PARAM DC.W 12,23 hace a PARAM una palabra que contiene 12 y PARAM+2 es una palabra que contiene 23
DS	"Declarar eSpacio": reserva un área de datos no inicializados (p. ej., STACK DS.B 100 establece un espacio de 100 bytes llamado STACK)
LLEN	Establece la longitud de línea
PAGE	Envía un salto de página durante el listado

```
Esto no es un comentario del todo pues falta el *
MOVE     D1, D2      *no debe haber espacios
                     *entre los operandos
MOVE     8TOIT,D5    *los nombres han de
                      comenzar
                     *con una letra y ADEMÁS
                      mediará
                     *al menos un espacio entre
                     *instrucción y operando
RTS                  *correcto
RTS      D1          *esto ya no (pues D1 aquí
                     *no tiene sentido)
*fin de ejemplo erróneo con comentarios verdaderos!
```

Digamos, para acabar, algo sobre la representación alfanumérica. Los números se representan en el ensamblador como valores decimales si no van precedidos del signo $. Así, 1234 es decimal pero $1234 es hexadecimal. Para las letras ASCII los caracteres se encierran entre comillas sencillas. Por ejemplo:

'Que pena, este curso se acaba!'

Acabamos de examinar el empleo del ensamblador como una herramienta de programación, ya no sólo como un medio de introducir codificación en el 68000. Las facilidades ofrecidas son bastante buenas especialmente con la facilidad macro y las bastante buenas directivas del ensamblador, así como los mensajes de error. Las ayudas de documentación también han sido dignas de resaltar, ¡dado que nunca han de olvidarse en favor de otras personas que sin duda habrán de leer los programas que usted escriba!



Perfecto en ralentí
El QL posee un microprocesador 68008, idéntico al 68000 salvo su bus de datos reducido a 8 bits. Ambos chips son compatibles por completo en cuanto a las instrucciones, por lo que nuestro curso de programación del 68000 es aplicable para programar el 68008. ¡La única diferencia está en la velocidad!

Facilidad MACRO

Justamente la facilidad MACRO es un método para sustituir textos en un programa fuente que se ha revelado como una herramienta poderosa de programación en grandes programas. Observemos que donde se especifica una "macro" se sustituirá el texto macro y el anteriormente definido se rechazará. Se puede, por ejemplo, definir así una macro ERROR:

```
ERROR    MACRO
         MOVE.B    #3,ERRORLOC
         JSR       SIGNAL
         ENDMACRO
```

Finalmente, la palabra ERROR producirá, en tiempo de ensamblaje, la siguiente serie de instrucciones:

```
MOVE.B   #3,ERROR.LOC
JSR      SIGNAL
```

Ahora podemos ampliar esta facilidad para incluir el empleo de parámetros a cada llamada si lo precisamos.

Mire este ejemplo:

```
         TTL       Macro ejemplo
         ORG       $1000
ADDON    MACRO
         ADD       \1,D2
         ADD       D2,\2
         ENDM

         ADDON     AVAL,SUM
         ADDON     BVAL,SUM
         ADDON     SUM,TOTAL
AVAL     DC.W      0
BVAL     DC.W      0
SUM      DC.W      0
TOTAL    DC.W      0
         END
```

Observará que\1 y\2 se refieren a los parámetros primero y segundo. En tiempo de ensamblaje, esto producirá:

```
3                ORG       $1000
4
5    ADDON       MACRO
6                ADD       \1,D2
7                ADD       D2,\2
8                ENDM
9
10               ADDON     AVAL.SUM
11+              ADD       AVAL,D2
12+              ADD       D2,SUM
13               ADDON     BVAL,SUM
14+              ADD       BVAL,D2
15+              ADD       D2,SUM
16               ADDON     SUM,TOTAL
17+              ADD       SUM,D2
18+              ADD       D2,TOTAL
19
20   AVAL        DC.W      0
21   BVAL        DC.W      0
22   SUM         DC.W      0
23   TOTAL       DC.W      0
24               END
```

Recuerde que los signos + después de los números de sentencia indican las líneas insertadas por el macroprocesador.