Mudar de sitio

Nos quedan por examinar cuatro modos de direccionamiento y mostrar sus diferencias con algunos ejemplos en código máquina

Para empezár hagamos un repaso de los modos que ya conocemos:

- *Direccionamiento absoluto*: En este modo el operando fuente o destino es una dirección de memoria.
- *Direccionamiento con registro*: La fuente o destino es un registro.
- *Direccionamiento indirecto con registro (y puntero)*: Apuntamos al objeto fuente o destino.
- *Direccionamiento indirecto con registro (y postincremento o predecremento)*: Recorremos una lista de datos de arriba abajo o viceversa.

Consideremos ahora el modo *inmediato*, en el que almacenamos una constante próxima a la instrucción.

- *Direccionamiento inmediato*: En este modo el operando fuente es una constante. Por ejemplo, MOVE.W #$43,D0 ordena que la constante (especificada mediante el símbolo #) $43 ($ significa hexadecimal) es llevada a D0. Se dice que es un direccionamiento inmediato porque la palabra constante se almacena en la propia instrucción MOVE. Es obvio que para operandos bytes la extensión toma un byte, y para palabras largas toma cuatro bytes.

Este modo inmediato es muy útil para establecer una constante, por ejemplo para un bucle con cuatro reiteraciones. Pero es de notar que pueden darse ocasiones en que sea mejor definir una constante en una posición determinada y emplear el direccionamiento absoluto hacia esta posición cada vez que se necesite. Por ejemplo:

```
MOVE    COUNT,D0
MOVE    COUNT,D5
COUNT   DC.W 4
```

Esto puede que se prefiera al direccionamiento en modo inmediato cada vez que se prevea que el valor COUNT ha de cambiar. Con el método absoluto cambiamos el contenido de COUNT; de otra manera tendríamos que investigar todo el código para todas las referencias en modo inmediato, empleando, por ejemplo, 4, lo cual, además de ser aburrido, puede comportar errores.

Es posible un modo inmediato *rápido* con ciertas instrucciones, donde la constante está contenida dentro de las instrucciones en una palabra. Tanto uno como el otro de estos dos ejemplos:

```
ADDQ    #3,D0
SUBQ    #1,D3
```

se codificarían en una palabra (y, por tanto, se ejecutarán más rápidamente ya que tienen la constante junto con la instrucción). Obsérvese, sin embargo, que las constantes con ADDQ y SUBQ sólo pueden estar en el intervalo #1 a #8.

La instrucción MOVE tiene también una versión rápida en la que el ámbito de las constantes se mueve entre −128 y +127. Así, MOVEQ #98,D4 pone en D4 el decimal 98.

Para reforzar su memoria, he aquí un ejemplo donde se emplean todos los modos de direccionamiento analizados hasta aquí:

```
1                LEA    OUTPUT,A1    Absoluto y registro
2                MOVE   A1,A2        Registro
3                MOVE   -(A1),D3     Predec. y registro
4                ADDQ   #3,D3        Inmediato rápido y registro
5                MOVE   D3,(A2)+     Registro y postinc.
6                ADD    #6,D3        Inmediato y registro
7                MOVE   D3,(A2)      Registro e indirecto
8
9       INPUT    DC.W   #6           Constante 6
10      OUTPUT   DS.W   2            Espacio para dos palabras
```

La posición INPUT contiene una constante de 6 y la dirección OUTPUT tiene un espacio para dos palabras.

Interesa tener en cuenta lo que está contenido en OUTPUT y OUTPUT+2 tras la ejecución de este fragmento en código máquina. Las líneas 1 y 2 hacen que A1 y 2 apunten a OUTPUT. La línea 3 hace que A1 apunte a INPUT antes de direccionar su operando, por lo que en D3 se cargará 6. La línea 4 añadirá 3 a D3, o sea el contenido de D3 es ahora 9.

En la línea 5, D3 se cargará en OUTPUT, ya que A2 apunta a OUTPUT. Una vez hecho esto, A2 se incrementa en dos para apuntar a la segunda palabra OUTPUT. La línea 6 suma 6 al contenido de D3 (contenido total, 15), que será cargado en la segunda palabra OUTPUT.

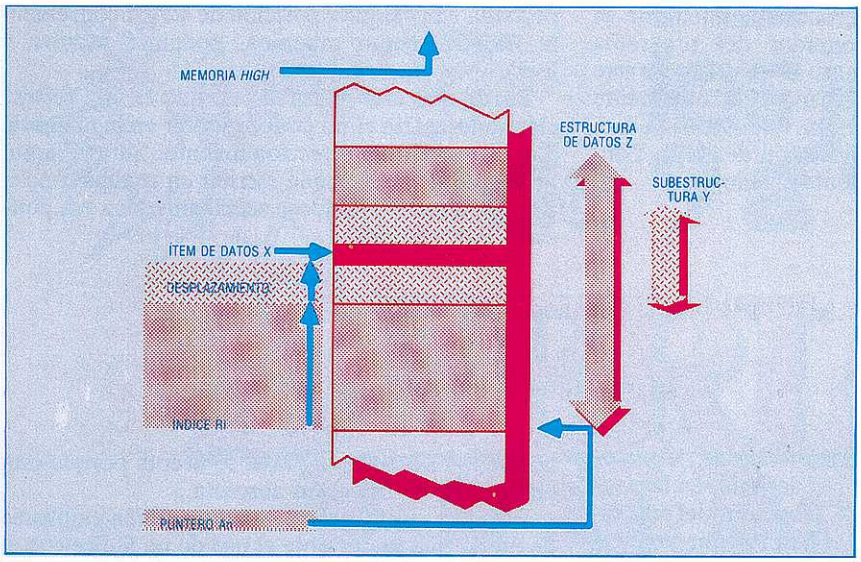
- *Direccionamiento indirecto (con desplazamiento e índice)*: Debemos considerar primero el significado que aquí se da a los términos *desplazamiento* (*displacement*) e *índice* (*index*). Con el 68000 por *desplazamiento* se entiende una "distancia" u *offset* fijo desde un punto de referencia básico; por su lado, *índice* significa un desplazamiento variable a través de un registro.

La diferencia entre estos dos términos queda ilustrada en el dibujo (página contigua). En él se muestran:

1) un componente de datos estructurados llamado Z, al que apunta el registro de direcciones An;
2) una subestructura interna llamada Y, indexada mediante un registro índice llamado Ri (registro de direcciones o de datos);
3) y un elemento de Y llamado X, al que se puede acceder con un desplazamiento fijo.

Ejemplos de estas estructuras de datos:

Saberse el camino
La *Guía del usuario* del 68000 contiene breves detalles de las instrucciones y modos del 68000, pero donde resulta más útil es en las sugerencias sobre cómo sacar provecho de las facilidades de éste en numerosas aplicaciones de ejemplo. El libro ha sido publicado por la editorial británica Sigma Press



Caroline Clayton

1) Z: tabla de registros, o tabla de dimensiones;
2) Y: el registro mismo, o bien una fila de una tabla;
3) X: elemento de un registro, o bien el entero de una tabla.

Se trata de tres componentes de datos estructurados que son conocidos en los modernos lenguajes de alto nivel tipo PASCAL y ADA, y, por tanto, es importante poder referirnos fácilmente a tales componentes. Naturalmente, no hay razón alguna por la que el programa ensamblador no pueda estructurar los datos de modo similar: ¡sin duda disponemos de medios para hacer esto con el 68000!

Volvamos una vez más a nuestro ejemplo de datos estructurados. Habrá notado que la dirección de X está formada por la suma An+Ri+desplazamiento, y que podemos alterar An y Ri conforme se ejecuta el programa. Podemos mirar ahora algunos ejemplos de direccionamiento, y comenzaremos con uno que lo ilustra cabalmente en su forma más sencilla (direccionamiento indirecto con desplazamiento):

```
MOVE.W DISP (A0),D1
```

donde DISP habrá sido definido previamente como nombre simbólico, supongamos, de 6. Esta instrucción emplea el direccionamiento indirecto con un desplazamiento de 6 como modo de direccionamiento fuente. Si, por ejemplo, A0 estuviera apuntando a la dirección $1000 (1000 hexa), entonces el contenido de la dirección $1006 se cargaría en D1. Nótese que al aceptarse una extensión de palabra entera con la instrucción para el desplazamiento, podemos tener un entero con signo de 16 bits completos para desplazamiento (valores entre +32767 y −32768).

Veamos ahora la modalidad de direccionamiento más poderosa que tiene el programador del 68000 (el desplazamiento), considerada en este ejemplo:

```
MOVE.W DISP (A0,D0.W),D1
```

En este caso, la dirección fuente está formada añadiendo juntamente el registro base, A0, con el registro índice, D0, y el desplazamiento fijo, DISP. Es de notar, sin embargo, que en este caso DISP sólo puede ser un entero con signo de ocho bits, aunque el registro índice puede ser una palabra larga de 32 bits completos.

Obsérvese que cuando usted compara estos dos modos de direccionamiento (el indirecto y el desplazamiento, con o sin índice), el desplazamiento siempre es fijo en tiempo de ensamblaje. Sin embargo, si usted necesita alterar dinámicamente un desplazamiento, entonces podrá optar siempre por usar:

```
MOVE.W 0(A0,D1),D2
```

donde D1 se convierte ahora en el desplazamiento de hecho (que podemos alterar en tiempo de ejecución del programa), y el desplazamiento fijo es cero.

Finalmente es oportuno señalar que el conjunto de instrucciones del 68000 ofrece estos dos potentes modos de direccionamiento para la mayoría de las instrucciones más comunes. Veamos un ejemplo de direccionamiento que emplea desplazamientos e índices, con bytes como atributos de datos.

```
1        LEA     LIST,A1
2        MOVEQ   #4,D1           D1:=4
3        MOVE.B  2(A1),D6        D6:=3
4        ADD.B   0(A1,D1),D6     D6:=8
5        MOVE.B  D6,4(A1,D1)     LIST+8:=8
6
7   LIST DC.B    1,2,3,4,5,6,7,8,0
```

A1 es puesto de modo que apunte a LIST en la línea 1 y seguidamente D1 es puesto a 4. En la línea 3, al puntero que hay en A1 se añade un desplazamiento de 2, de forma que el tercer elemento de LIST es cargado en D6. Después, con un índice de 4 en D1, se añade a D6 el contenido de elemento 5, siendo copiado en LIST+8.

Obsérvese que en la línea 4 hemos empleado el desplazamiento de 0: esto permite emplear D1 como registro índice, que podemos alterar cuando se ejecuta el programa, si fuera el caso.

• *Direccionamiento relativo al PC*: Antes de pasar a analizar en detalle los modos de direccionamiento relativo al PC debemos echar un vistazo a algunas directivas del ensamblador. Las directivas son instrucciones dadas al ensamblador que no producen directamente código ejecutable, pero que influyen en factores tales como el formato del listado del programa fuente o la definición de símbolos. Una de estas directivas se refiere específicamente a la dirección de inicio del programa y el tipo de código producido. Es la directiva ORG, o sea origen.

Normalmente se especificará la dirección de inicio, por ejemplo así:

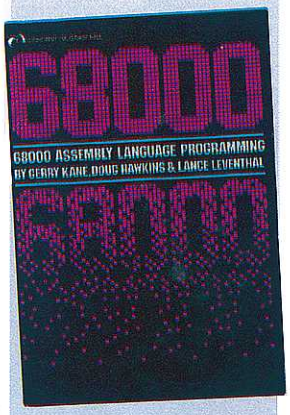
```
ORG    $1000
```

Con ello establecemos la dirección de inicio en $1000 (en realidad es la dirección de carga para el cargador binario), y todo el código que sigue es cargado en direcciones secuencialmente crecientes. Si aparece una nueva directiva ORG, establecerá entonces una nueva dirección de carga en ese punto. Una variante de esto sería:

```
ORG.L    $2000
```

donde todas las referencias posteriores se tomarán como palabras largas completas, y por ende cualquiera de estas referencias ocupará dos palabras enteras.

La directiva RORG define una dirección de carga



Enseñanza general
Este libro sobre programación en lenguaje assembly del 68000 es una guía completa sobre el tema publicado por la editorial británica Osborne/McGraw-Hill. Los programadores experimentados encontrarán algo superfluas las secciones sobre teoría de la programación y el estilo del libro es enfáticamente magistral. Pero si por un lado le falta informalidad por otro incluye toda la información que usted pueda necesitar. Un buen detalle es el empleo de encabezamientos de párrafo en mayúsculas, lo que permite buscar rápidamente un tema concreto a través de las secciones

desde la cual todas las referencias de memoria se consideran relativas al contador del programa (PC), dejando de ser absolutas. El principio de este tipo de código se basa en que, independientemente de donde se cargue el código, las referencias de memoria corresponden a esa dirección de carga. Pongamos por ejemplo el siguiente listado:

```
                        RORG    $2000
2000  D47A      START   ADD     CONST1,D2
2002  000A
2004  3202              MOVE    D2,D1
2006  6000              BRA     START
2008  FFFB
200A  4E40              TRAP    #0
200C  0026      CONST1  DC.W    38
```

Aquí, CONST1 está referenciado con un desplazamiento de 16 bits de A hexa (10 decimal) en la posición 2002, que se añade al PC (contador del programa). Para cuando se cargue el desplazamiento en el ejemplo, el PC será 2002 de modo que la dirección de datos fuente será 200C. Obsérvese que no hay instrucciones específicas o diferencias en la sintaxis de direccionamiento para obtener el código relativo al PC; todo lo que se requiere es la directiva RORG del ensamblador.

Hay unos cuantos puntos más sobre este código dignos de nota. Primero, que la instrucción BRA (*branch always*: bifurcar siempre) tiene un desplazamiento asociado con ella en la dirección 2008, el cual, una vez añadido al PC, dará la dirección de operando START. Esto significa que la instrucción BRA dará siempre el código relativo al PC. El segundo punto a notar es la instrucción TRAP. Se usa en este contexto como una instrucción de parada, con el añadido de que la entrada se hace a una pantalla que imprime los registros y permite al usuario el examen de la memoria. Esta instrucción, si su pantalla se lo permite, puede resultar muy útil.

Veamos otro ejemplo, donde el direccionamiento de modo relativo al PC da un desplazamiento negativo:

```
                        RORG    $2000
2000  0026      CONST1  DC.W    38
2002  DA7A      START   ADD     CONST1,D2
2004  FFFC
2006  3202              MOVE    D2,D1
2008  6000              BRA     START
200A  FFF8
200C  4E40              TRAP    #0
```

Aquí, la referencia a CONST1 es un desplazamiento negativo en la dirección #2004. (Naturalmente, usted conocerá que FFFC es negativo porque el bit de signo estará activado. La magnitud, o tamaño, de los números pueden encontrarse invirtiéndolos y añadiendo uno; por tanto, FFFC es el decimal −6).

Una limitación a este direccionamiento es que solamente se permite como operando fuente. Por ejemplo:

```
        RORG    $2000
TOTAL   DC.W    0
START   ADD     D2,TOTAL
```

generará un error de ensamblador. Esto significa que este tipo de direccionamiento es muy conveniente para el código que ha de ponerse en una ROM (dado que no podemos escribir en las ROM) en una dirección fija, pero todavía puede ser comprobada en cualquier posición de RAM conveniente (donde, como sabemos, podemos escribir y leer).

Puede que esté pensando en que es una restricción innecesaria el no poder escribir en la memoria con este modo de direccionamiento. Sin embargo, observe que usted puede escribir en cualquier posición absoluta refiriéndose sencillamente a esa posición. Por ejemplo:

```
        ORG     $1000
TOTAL   DS.W    0
        RORG    $2000
CONST1  DC.W    38
START   ADD     CONST1,D2
        MOVE    D2,TOTAL
```

donde la referencia a TOTAL se acepta porque está en un área de dirección absoluta.

El modo relativo al PC puede también empliarse de modo que sea posible el uso de un índice con el desplazamiento.

Por ejemplo:

```
        RORG    $3000
INDEX   DC.W    10
START   MOVEA   INDEX,A5
        ADD     6(A5),D2
```

En este caso tenemos que usar la instrucción MOVEA, que carga A5 con el contenido de la posición de memoria INDEX. No podríamos usar LEA porque tomaría la dirección de INDEX (de ningún modo permisible en el modo relativo al PC). Es claro que una instrucción MOVE directa sería ilegal, como ya hemos visto (un registro de dirección no es legal como operando destino).

Volvamos ahora al ejemplo donde el operando fuente de la instrucción ADD será el valor de PC después de la instrucción ADD más el registro índice (aquí, A5) con un desplazamiento de 6. En este ejemplo, la dirección fuente será 16 bytes más allá de la dirección que contiene la palabra de extensión 6.

Una limitación de este modo es, sin embargo, el que el desplazamiento esté formado por ocho bits dentro del opcode, limitando nuestro desplazamiento a un número de bytes desde +127 a −128.

La importancia de este modo relativo al PC es que el código puede ejecutarse en cualquier sitio de la memoria, como hemos visto en el caso de escribir código para la ROM. Obsérvese, sin embargo, que esta forma de direccionamiento es extremadamente útil, por lo general, para escribir código independiente de la posición en memoria. Esta forma de código puede ser necesaria para extensos programas de muchos módulos donde la posición de un módulo en la memoria no es fijada hasta que el módulo se carga en la memoria. Naturalmente, para largos sistemas de multiprogramación, puede que necesitemos una unidad de gestión de la memoria con facilidades adicionales (que proporciona Motorola), pero para cualquier esquema de memoria más sencillo el modo relativo al PC es muy importante.

- *Direccionamiento implícito*: Este modo de direccionamiento no debería ser dificultoso, dado que el opcode especifica los registros que hay que usar. Por ejemplo, la instrucción RTS afectará al PC y al puntero de la pila; BRA afecta al PC.