La dirección exacta, por favor

Investigaremos los modos de direccionamiento del 68000. Esto nos ayudará a la hora de programar el chip con registros y tablas

En el capítulo de introducción de esta serie hablamos de la capacidad de direccionamiento del 68000 a propósito del juego de instrucciones. En particular advertimos que a pesar de tener un amplio abanico de modos de direccionamiento con los que es fácil referenciar bytes, palabras y palabras largas, debemos ser muy cautos en el empleo de tales modos con determinadas instrucciones.

Sea la siguiente *instrucción generalizada*:

```
OPCODE fuente, destino
```

Ésta es una manera semiformal de describir lo que realiza una clase de instrucciones con los operandos fuente y destino. La secuencia puede que exija tomar el operando fuente (dondequiera que se encuentre, y a través del cálculo que sea preciso para llegar hasta él), realizar la operación definida por el opcode sobre el operando y depositar el resultado en el operando destino (aquí también puede que sea preciso realizar algún cálculo para obtener la dirección de este operando). Por ejemplo, la instrucción MOVEA D3,A6 hace que el contenido de D3 (fuente) se lleve a A6 (destino). El opcode, MOVEA, indica que ha de moverse una dirección.

Se trata de un ejemplo muy sencillo de direccionamiento, donde no es preciso cálculo alguno para obtener la dirección de los operandos. En el otro extremo podríamos encontrarnos con que uno de los operandos esté direccionado mediante la suma del contenido de un registro de direcciones y un desplazamiento entero más el contenido de un registro índice (similar a la instrucción LD r, (IX+d) del Z80). Ya tendremos tiempo de comentar esto más adelante; de momento, baste con advertir que puede haber un buen puñado de operaciones aritméticas por realizar hasta dar con la dirección de los operandos.

Volviendo a nuestro modelo generalizado de instrucciones, también es posible que el opcode precise tan sólo un operando, como en este caso:

```
OPCODE fuente
```

Por ejemplo, la instrucción de bifurcación (como BRA BACKHERE) sólo necesita un operando (es decir, la dirección para bifurcar hacia BACKHERE). Finalmente, podemos también encontrarnos sólo con:

```
OPCODE
```

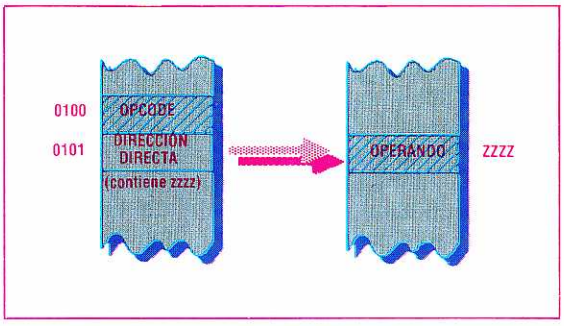
sin operando alguno. Un ejemplo típico de esta forma es la instrucción NOP, que sirve para indicar que no se realice ninguna operación. En efecto, se trata de una instrucción ficticia, y su utilidad es evidente en los parcheos manuales o en el cambio de programa en la memoria también manual.

Al emplear este modelo generalizado de la gama de instrucciones lo importante es tener en cuenta que siempre que haya operandos habrá algún tipo de cálculo de direcciones. Este cálculo es el que ha de ser especificado por el programador entre el conjunto de los cálculos o los modos de direccionamiento disponibles en el 68000.

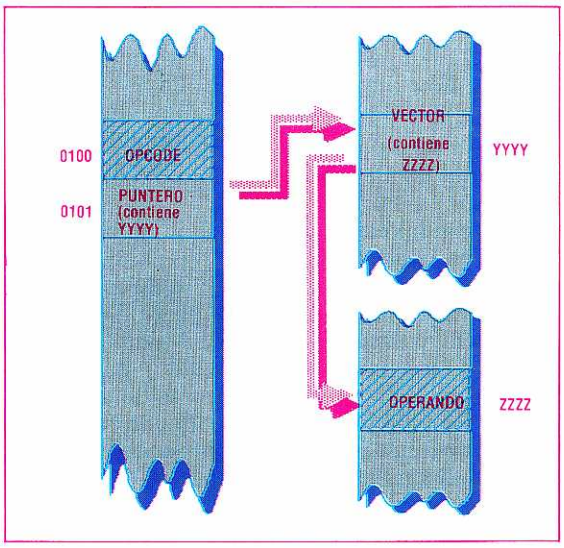
Muchos ordenadores disponen de al menos cinco *modos de direccionamiento* o maneras diferentes de direccionar los operandos. Nuestros diagramas ilustran la diferencia operativa de dos de estos modos:

- *Direccionamiento directo (o absoluto)*: En este modo la dirección de memoria del operando se almacena en la propia instrucción.
- *Direccionamiento indirecto (o puntero)*: En la instrucción se da una dirección de memoria que contiene a su vez la dirección del operando.

Como muestra el dibujo, en el direccionamiento directo el opcode opera sobre el operando que está en la dirección XXXX, mientras que en el direccionamiento indirecto, el operando será hallado en la po-



Direccionamiento directo
En este modo de direccionamiento el opcode (código de operación) de la instrucción es seguido inmediatamente por la dirección de una posición de memoria que contiene los datos del operando



Direccionamiento indirecto
El direccionamiento indirecto requiere que la operación especificada por el opcode acceda a su operando mediante una dirección "vector". La dirección del vector sigue inmediatamente al opcode, y el vector contiene la dirección del operando requerido. Este modo es muy útil cuando se calcula la dirección de un operando en tiempo de ejecución, dado que sólo se necesita calcular de nuevo el contenido del vector

sición ZZZZ, a la cual apunta un *puntero* que en este caso se encuentra en la posición YYYY de la memoria. Los otros tres modos principales de direccionamiento son:

- *Modo inmediato*: Cuando uno de los operandos de la instrucción es una constante. Por ejemplo, en la instrucción MOVEQ #25,D3 la constante 25 está direccionada en modo inmediato.
- *Modo de registros*: En este modo, el operando es uno de los registros disponibles y es especificado en el mismo código de la instrucción. Los operandos en la siguiente instrucción MOVE D2,D4 son los registros de datos D2 y D4.
- *Modo implícito*: Aquí los operandos están implícitos en la misma instrucción. Por ejemplo, en el caso de RTS (*ReTurn from Subroutine*: retorno de subrutina) son operandos implícitos el puntero de la pila y el contador del programa.

Observemos con mayor atención la manera en que el 68000 direcciona sus operandos. Además de los modos generales de direccionamiento que acabamos de ver, el 68000 tiene un *modo relativo de contador de programa* (también llamado *PC relativo*). Examinemos estos modos uno por uno:

- *Direccionamiento absoluto*: Empleando este modo podemos acceder a cualquier posición de la memoria. La dirección del operando aparece después de la instrucción. Por ejemplo:

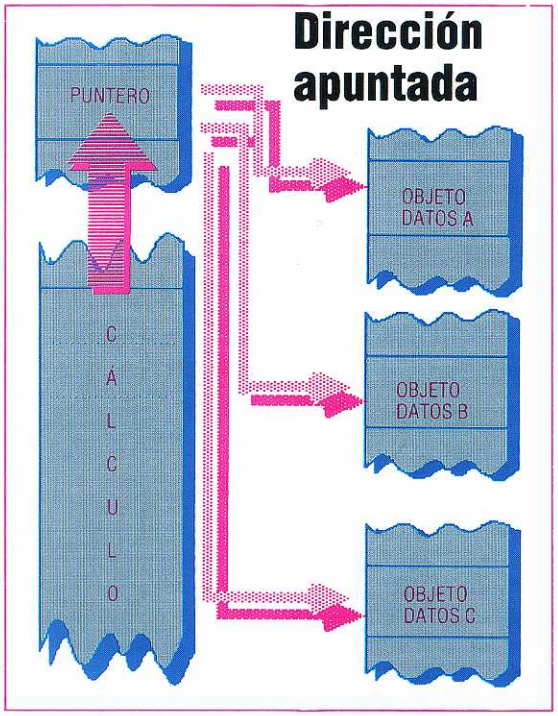
```
Direc.:   Código:  Etiq.:   Instrucción:
1000      D678             ADD     DATA,D3
1002      2000
.
.
.
2000      0001     DATA    DC.W    1
```

En este ejemplo podemos ver que el nombre simbólico DATA ha recibido la dirección $2000, la instrucción "suma (ADD) la fuente absoluta (DATA) al destino D3" ha sido codificada como D678, y que DATA de dirección absoluta se alberga en la posición $1002 (llamada *extensión de la palabra*). Otro ejemplo donde sólo encontramos un operando es:

```
Dir.:     Código:  Etiq.:   Instrucción:
1000      4278             CLR     COUNT
1002      3000
.
.
.
3000      0        COUNT   DS      1
```

Aquí el contenido de la posición $3000 (COUNT) queda limpio (puesto a cero) tras la ejecución de la instrucción CLR (*clear*: limpiar).

En el capítulo anterior dijimos que el PC era un registro de 32 bits (aunque sólo 24 de estos bits son significativos). Esto significa que la dirección absoluta que especifica el operando puede constar de más de una palabra como en los dos ejemplos anteriores. ¿Cómo hace el ensamblador para saber cuánto espacio ha de reservar para la dirección absoluta? Sin duda sería un dispendio injustificado el tener una extensión de palabra larga por cada referencia de dirección absoluta, por ello lo que ocurre es que se emplea la extensión adecuada siempre que se conoce la dirección del operando (en el caso de una referencia hacia atrás). En otras circunstancias habremos de especificar al ensamblador el empleo de las extensiones de palabra corta o palabra larga.



Volvamos al ejemplo de ADD para mostrar los efectos de una extensión de palabra larga:

```
Dir.:     Código:  Etiq.:   Instrucción:
1000      D679     ADD     HIDATA,D3
1002      0020
1004      0000
```

En este ejemplo la dirección absoluta de HIDATA es $200000. Nótese que el código para la parte ADD de la instrucción sigue siendo D6 pero que la parte de la dirección del operando ha cambiado de $78 hasta $79. En próximos capítulos veremos las formas para lograr esta extensión de palabra larga para el direccionamiento absoluto.

- *Direccionamiento por registro*: Es la manera más sencilla de direccionamiento en el 68000; en este caso el operando es uno de los registros del microprocesador. Por ejemplo, ADD D0,D3, donde la palabra D0 se añade al contenido de D3.

Hay algunas limitaciones en el empleo de este modo. Por ejemplo, no es posible tener un registro de dirección como destino de la instrucción ADD: así, no se acepta ADD D0,A4. Esto tiene arreglo si nos valemos de una instrucción diferente, ADDA D0,A4 donde ADDA es la instrucción de dirección de suma.

Si deseamos emplear palabras largas como objetos de datos en los ejemplos anteriores, deberemos incluir el atributo .L con la instrucción: ADD.L D0, D3 empleará las palabras de 32 bits enteras como objetos de datos.

- *Direccionamiento indirecto por registro*: Este modo es probablemente el más importante del 68000, dado que proporciona el *puntero* mencionado anteriormente y también los medios con los que se ejecutan las operaciones sobre la pila. Analicemos primero el empleo del puntero.

En la tarea de la programación necesitamos con frecuencia apuntar a un objeto de datos, ya sea un byte, una palabra larga o un objeto estructurado de datos como un registro o una tabla. Puede que, entonces, deseemos repetir el cálculo o la operación en otro miembro del mismo tipo de objeto de datos: es aquí donde el puntero resulta útil. El dibujo del puntero que adjuntamos muestra cómo puede ser empleado para dirigirse a diferentes elementos de un registro. Inicialmente el puntero dirige el cálculo que se ha de efectuar sobre el objeto de datos A; el puntero puede entonces ser restaurado para dirigir el cálculo que ha de efectuarse sobre cualquiera de los restantes objetos de datos.

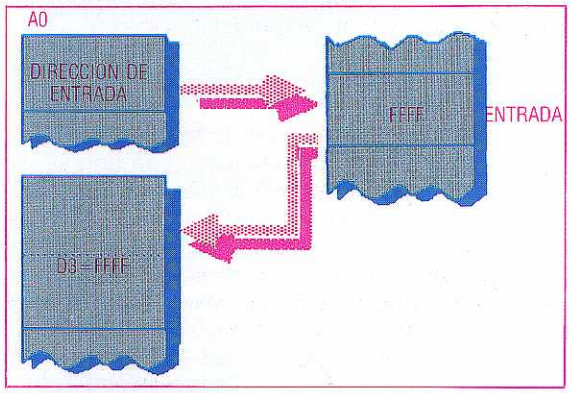
El elemento del 68000 a utilizar para lograr esto es un puntero de registro de direcciones, en vez del puntero almacenado junto a la instrucción, como en nuestro ejemplo general de modos de direccionamiento.

Esto puede comportar ligeros inconvenientes en algunas ocasiones, pero disponemos de ocho registros de direcciones.

Veamos ahora un ejemplo sencillo de modo de direccionamiento indirecto:

```
LEA      INPUT,A0
MOVE.W   (A0),D3
```

La instrucción LEA carga A0 con la dirección del objeto de datos de entrada llamado INPUT, el cual es seguidamente copiado en D3, como se muestra en el siguiente esquema:



Veamos ahora cómo se amplía este modo para operar sobre listas de datos. Ante todo, tenemos la *extensión de postincremento*. Se trata de que, una vez que se ha accedido al objeto de datos mediante el puntero, éste se incrementa para que apunte al contenido siguiente de la lista.

Examinemos el empleo de la extensión de postincremento en un ejemplo donde los objetos de datos son palabras:

```
MOVE.W   (A0)+,D3
```

Aquí el puntero en A0 es incrementado en dos unidades después que se ha accedido a la palabra apuntada por A0 y copiada en D3. Así, cuando apuntamos a la dirección $2000 originalmente, después de la operación MOVE.W veremos que A0 contiene $2002.

Es claro que si hemos empleado objetos de bytes entonces una operación MOVE.B sólo incrementará A0 en una unidad; y en cuatro con la operación MOVE.L.

La potencia de este modo de direccionamiento es obvia si se emplea en un bucle de programa para realizar algún cálculo en cada componente de una lista, por ejemplo. Así:

```
LEA      INPUT,A0      Inicialización
MOVE.W   (A0),D3       Copia cada
                         componente
                :
                :
         Cálculo
                :
                :
         Comprueba fin
           de la lista
```

Cada vez que se realiza el cálculo se apunta automáticamente al componente siguiente de la lista tras cada ejecución de la instrucción MOVE.

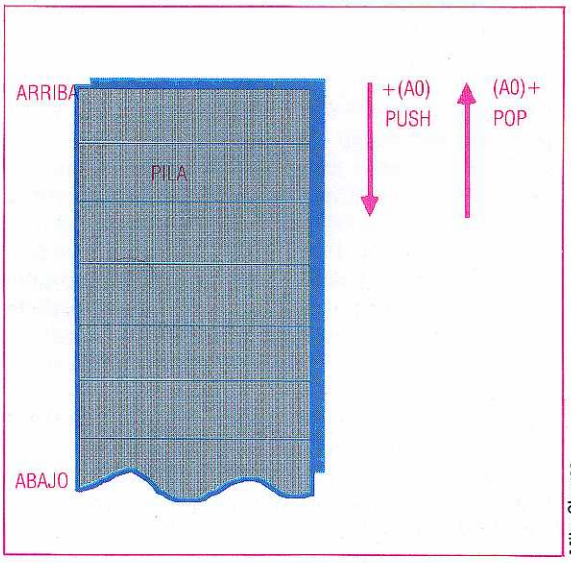
La otra extensión es denominada *predecremento*. En este caso el puntero de direcciones se decrementa antes de acceder al objeto de datos. P. ej.:

```
MOVE.W   -(A0),D3
```

En este caso A0 se decrementará en dos unidades antes de ser empleado para copiar en D3 el objeto de datos.

El direccionamiento indirecto predecremental es el modo complementario del direccionamiento indirecto incremental. El posdecremento progresa a través de una lista a medida que se incrementan las direcciones, mientras que el predecremento funciona al revés. Sin embargo, no podemos separar las operaciones de *pre* y *post* según nos convenga. Por ejemplo no son admisibles ni (A0)- ni +(A0). Debemos, en cambio, respetar el predecremento -(A0) y el posdecremento (A0)+ especificados.

Habrá notado que en cierto sentido los modos de direccionamiento *pre* y *post* son operaciones de pila. Nos servimos de estas operaciones para "poner en" o "sacar de" una pila, y una pila es, como se define convencionalmente, una serie de direcciones de arriba hacia abajo. De esta manera, la instrucción MOVE D0,-(A0) "pone" (*push*), y MOVE (A0)+,D0 es una operación que "saca" (*pop*), tal como se aprecia en el siguiente dibujo.



Mike Clowes

Dirección apilada
Las instrucciones PUSH y POP, tan conocidas de los programadores del Z80, pueden simularse mediante las potentes extensiones predecremento y posdecremento del 68000, que pueden usarse para ejecutar un PUSH y un POP respectivamente

Más adelante estudiaremos las pilas y cómo se emplean en el 68000. De momento, baste con observar que disponemos de un modo de direccionamiento muy cómodo para acceder a listas de datos.