Siga las instrucciones

Iniciamos aquí el estudio detallado de las instrucciones del 68000 de Motorola. Empezaremos por las más sencillas

En los dos capítulos anteriores hemos examinado la forma en que el 68000 direcciona sus operandos y hemos mostrado el empleo de algunas instrucciones (tales como MOVE, ADD y la generalización 'OPCODE'). Demos un paso más para estudiar el conjunto de instrucciones con más detalle, comenzando por las instrucciones de copia de datos, para pasar después a las instrucciones de cálculo que conciernen a la aritmética en sistema de numeración binario.

Vamos a demostrar la utilidad de instrucciones más importantes, y a resaltar todo detalle relevante o posible trampa en su empleo. Si usted pretende programar el 68000 con frecuencia, habrá de considerar, sin embargo, la adquisición o bien del *Manual del usuario del 68000 de Motorola* o bien alguna de las restantes publicaciones que mencionamos en este curso.

Antes de estudiar las instrucciones hemos de examinar con algún detalle el contenido del registro de estado (SR: *status register*). La mitad de este registro contiene los códigos de condición que indican el resultado de la última instrucción ejecutada. Cada código de condición puede ser considerado como memoria de un bit asignada a una condición aritmética particular. Examinémoslos uno por uno:

- *BIT 0: bit de arrastre o bit C:* Este bit se pone a uno cuando la operación aritmética produce un arrastre en el bit más significativo del operando de datos. Por ejemplo:

la suma de	0110 0000
y	1110 0000
da	1 0100 0000

En este caso, se arrastra un 1 del bit más significativo de la suma, que tiene un byte de longitud.

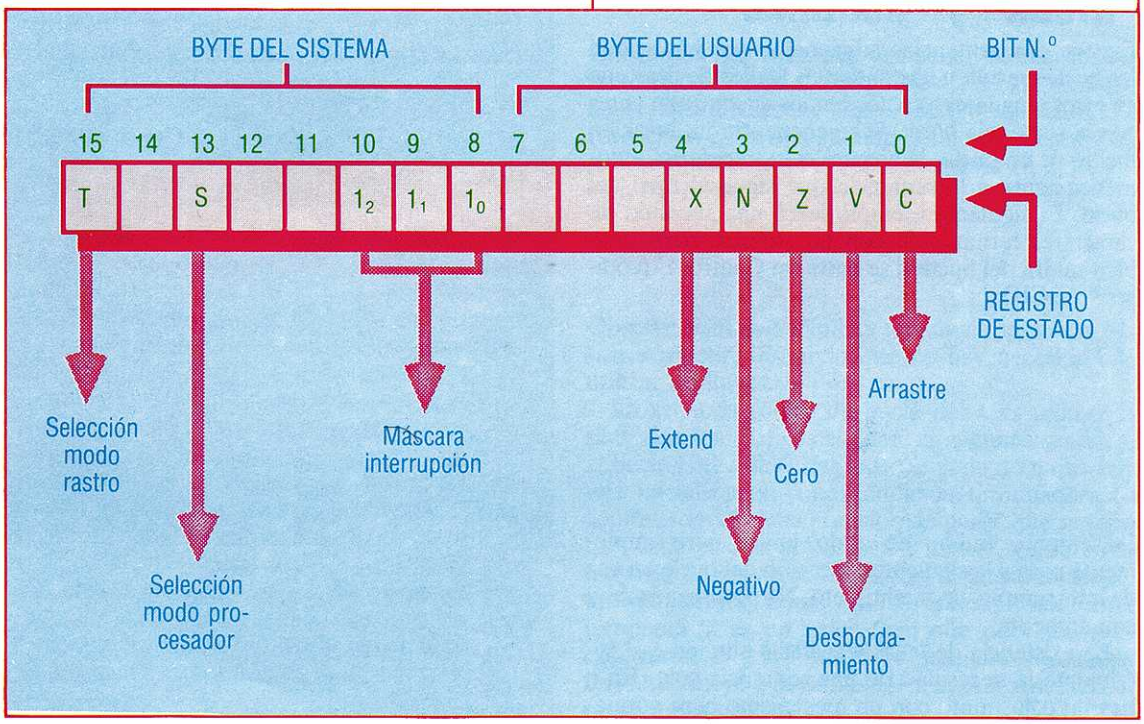
En este ejemplo el arrastre es llevado al bit C del SR, y no al bit menos significativo de la siguiente unidad de datos (en este caso, una palabra). Es de notar también que el arrastre puede ser significativo o no, según lo que se esté haciendo. Si, por ejemplo, se está calculando algún resultado de precisión múltiple (sobre otras unidades del operando de datos), es claro que el bit C debe ser significativo.

- *BIT 1: bit de desbordamiento o bit V:* Se activa cuando el resultado del cálculo no se ajusta al inter-

Estado mayor

El registro de estado de 32 bits se divide en las secciones del sistema y del usuario, cuyo significado de bits se muestra aquí. Obsérvense los bits de estado empleados para señalar mediante flag los modos del procesador: el 68000 puede operar en dos modos, el supervisor y el usuario. La razón de esto es que en un entorno de multiproceso es necesario ser capaz de controlar las diferentes tareas que se ejecutan. Por lo general, las tareas se ejecutan en el modo usuario (que posee su propio puntero de pila y no permite la alteración de los bits de estado del sistema), y el software de sistema (como sería la supervisión de las tareas) se ejecuta en el modo supervisor.

Para comenzar, es frecuente que usted entienda más conveniente el empleo del modo supervisor del 68000, en el cual son legales todos los opcodes, y desentenderse del problema de evitar las instrucciones privilegiadas en el modo usuario. Además, el 68000 posee un modo *trace* (rastro), que es una facilidad muy eficaz a la hora de depurar errores y que permite al usuario rastrear los pasos de las instrucciones, mientras que el 68000 llama automáticamente a la rutina de depuración para el usuario tras cada instrucción



valo de bits de los operandos de datos. Por ejemplo, si se suma un 1 a 32767 (el número entero positivo máximo para una palabra de 16 bits), se obtiene un desbordamiento en los operandos para datos de palabras, y el resultado binario carecerá de significado.

- *BIT 2: bit cero o bit 2:* Se activa cuando el resultado de un cálculo previo es cero.
- *BIT 3: bit negativo o bit N:* Se activa en los resultados negativos.
- *BIT 4: bit de extensión o bit X:* Se emplea en operaciones de multiprecisión, pero en general equivale al bit C (aunque no viene afectado por las instrucciones MOVE).

La instrucción del 68000 para copiar datos es MOVE, que copia de una fuente a un destino. Es una instrucción versátil: se puede emplear con ella cualquier modo de direccionamiento como fuente y casi la mayoría de los modos de direccionamiento como destino (a excepción de los registros de direcciones, los modos relativos al PC y el modo inmediato). Este grupo de modos de direccionamiento se conoce como *modos de datos alterables,* y existe un subgrupo dentro de éste denominado *modos de memoria alterable* (datos alterables menos registros de datos). De ambos grupos nos ocuparemos más adelante.

Volviendo a cómo se ha de emplear la instrucción MOVE, se advertirá que ninguna de las instrucciones MOVE siguientes es lícita:

```
RORG $1000
MOVE D2,MIMI    el PC relativo a MIMI no es válido
MOVE D2,A2      no es válido el registro de
                direcciones como destino
```

Nótese que la instrucción MOVE afecta sólo a los bits N y Z del SR, y que los bits V y C serán puestos a cero.

Para solventar el problema de los registros de direcciones como operandos de destino, se dispone de dos opciones:

- Emplear MOVEA, que toma el contenido del operando fuente y lo copia en el registro destino de direcciones.
- Emplear LEA, que toma la dirección fuente (por lo general, absoluta) y la copia en el registro de direcciones.

Ninguna de estas dos instrucciones afectará a los códigos de condición. Del mismo modo, hay instrucciones especiales para llevar los datos desde y hacia el registro de estado y el puntero de pila del usuario, pero son instrucciones que tienen que ver sobre todo con la programación de sistemas.

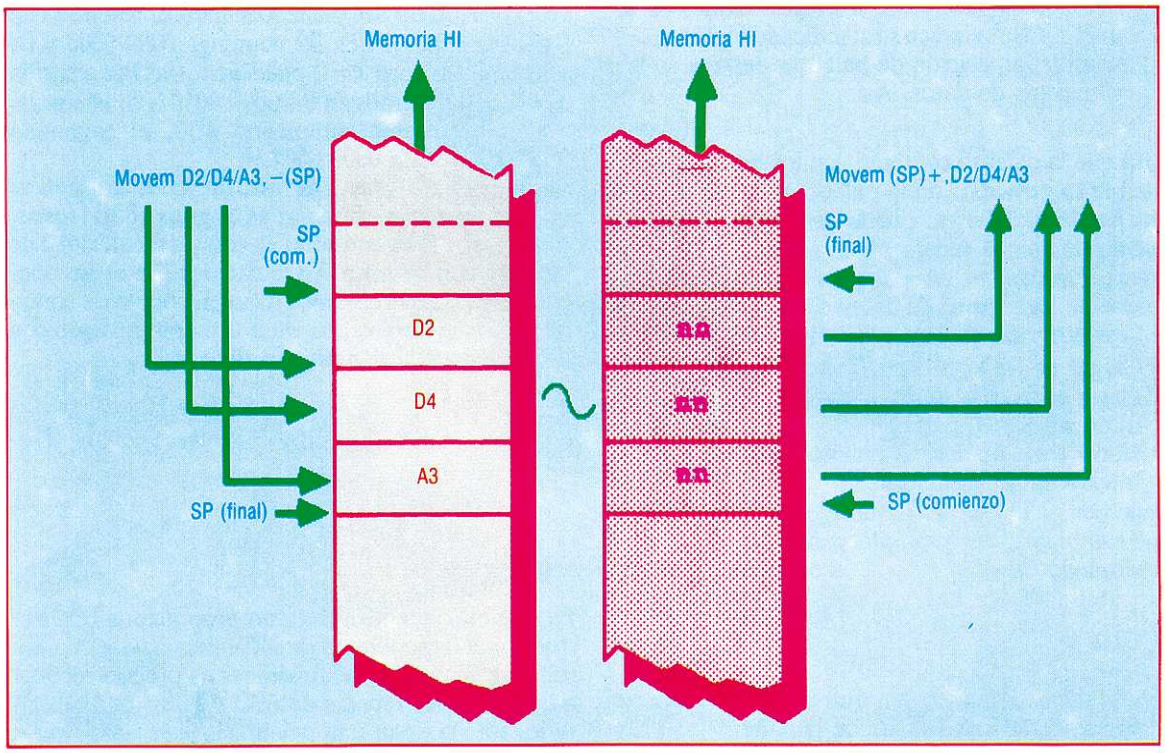
Otra instrucción para copiar datos extraordinariamente poderosa es MOVEM, que nos permite guardar o recuperar cualquier registro declarado (de direcciones o de datos) desde o hacia posiciones de memoria consecutivas. Esto significa que al entrar en una subrutina todos los registros que de otra manera perderían su validez en ella pueden preservarse y a la salida recuperarse. Por ejemplo:

```
entrada MOVE     D2/D4/A3, PAD
                 (el código de la subrutina
                 emplea D2, D4 y A3)
salida MOVEM     PAD, D2/D4/A3
```

Con ello se guardarán D2, D4 y A3 en PAD a la entrada, y los tres registros se restaurarán a la salida. Los registros se pueden guardar también en la pila. Así, por ejemplo, podríamos tener:

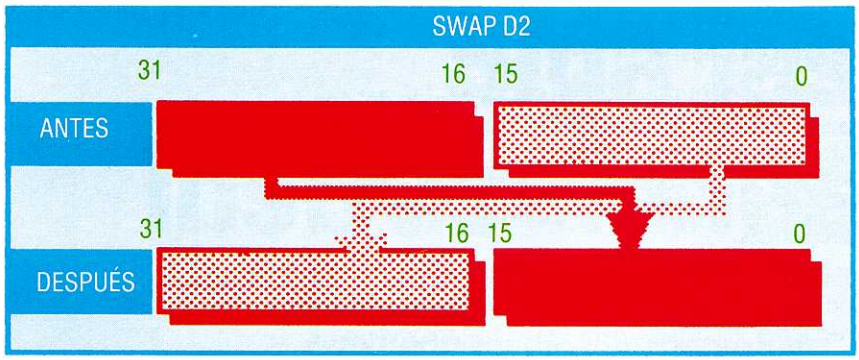
```
entrada MOVEM    D2/D4/A3,-(SP)
                 código de la subrutina
salida MOVEM     (SP)+,D2/D4/A3
```

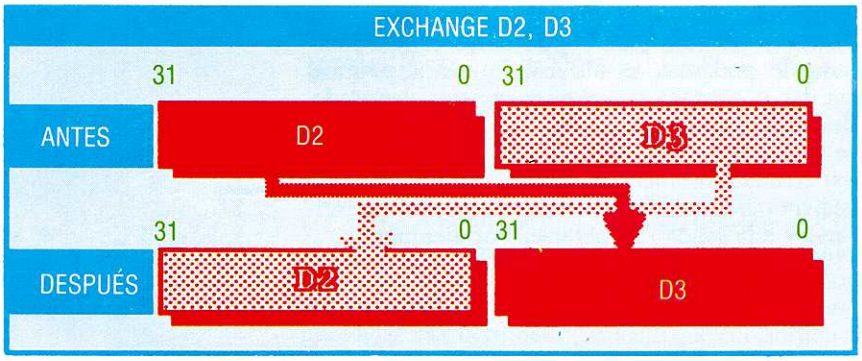
Otra variante de la instrucción MOVE es la instrucción rápida *(quick)* MOVE (o MOVEQ). Es útil cuando se establecen constantes de ocho bits con signo (desde +127 hasta –128) en un registro de datos dentro de una palabra de memoria. Entre los em-



Caroline Clayton

Vayan pasando, por favor
La poderosa instrucción MOVEM nos permite manipular los "listados de registros" en secuencia dentro de una instrucción. Mostramos aquí su uso junto con las instrucciones predecremento y postincremento para poner y hacer salir tres registros hacia y desde la pila. La posición del puntero de la pila es la que se muestra antes y después de cada instrucción





Dos tipos de intercambio
El empleo del almacenamiento de palabras largas puede resultar más bien pesado cuando son necesarios operandos con longitud de un byte o una palabra. SWAP nos ayuda en este problema intercambiando los valores contenidos en los bits del 0 al 15 de un registro con los contenidos en los bits del 16 al 32. Además, EXG (la instrucción de intercambio) permite el intercambio de palabras largas, como se muestra en el esquema inferior

pleos más comunes estará el establecer contadores de bucles dentro de un registro de datos. Por ejemplo:

```
MOVEQ #34,D2
```

establecería 34 en D2 en una palabra de memoria. Nótese que si se quita la Q algunos ensambladores no asumen el modo rápido. Por esto, la instrucción MOVE #34,D2 se codificaría en dos palabras.

Otra instrucción para copiar datos de posible utilidad para operaciones de la pila es PEA *(push effective address:* poner la dirección efectiva... en la pila). Por ejemplo, PEA BETTY llevará la dirección de BETTY a la pila: hará –(SP).

Examinemos finalmente las instrucciones de intercambio. Como indica su significado inglés, SWAP intercambia la posición de palabras enteras dentro de un registro de datos. Así:

```
SWAP  D2
```

hará que la palabra contenida en los bits del 0 al 15 cambie su posición con la palabra contenida en los bits del 16 al 31. Esto puede ser útil si por un supuesto deseamos repetir algún cálculo sobre palabras almacenadas en palabras largas completas dentro de un registro de datos. En este caso el procedimiento sería el siguiente:

- Cargar en D2 la palabra larga completa
- Hacer el cálculo sobre la palabra
- Intercambiar D2
- Hacer cálculo sobre la palabra
- Intercambiar la palabra

Tenemos también la instrucción de cambio EXG, que cambia palabras completas de 32 bits según un determinado modelo. Algunos ejemplos:

```
EXG D2,D3   Intercambio entre registros de datos
EXG A3,A4   Intercambio entre registros de dirs.
EXG D2,A5   Intercambio de datos y direcciones
```

Esta instrucción no es más que una SWAP para palabras completas de 32 bits de tamaño.

Aritmética con enteros

Estas instrucciones forman la base de todos los cálculos aritméticos (ya sean con fracciones, valores reales o de doble precisión) que, básicamente, consistan en sumar números binarios. Aun cuando la aplicación que usted desea no implique cálculo numérico alguno, necesita saber cómo realizar las operaciones aritméticas más sencillas a fin de que, por ejemplo, pueda convertir códigos de caracteres o formar índices de tablas.

La instrucción ADD se limita a sumar la fuente con el destino y almacenar el resultado en el destino. Los objetos de datos pueden ser de cualesquiera tamaños de atributos de datos y son afectados todos los códigos de condición. Por ejemplo, ADD.W D2,D3 sumará el contenido de palabra de D2 con D3 y almacenará el resultado en D3.

Para los datos fuente se pueden emplear todos los modos de direccionamiento, pero es necesario el empleo de una instrucción especial, ADDA, cuando el destino es un registro de direcciones. Así, ADDA D2,A4 sumará el contenido de D2 en A4.

Para el caso de datos inmediatos, está también la instrucción ADDI. Esta instrucción suma el dato inmediato (se permiten todos los atributos) almacenado como una palabra de extensión en el destino (se permiten sólo los modos alterables de datos). Así, ADDI #3423, BETTY sumará 3423 al contenido de BETTY.

A semejanza de MOVE, también para ADDI disponemos de una forma rápida, ADDQ. Pero el empleo de esta última instrucción sólo admite datos dentro del intervalo del 1 al 8. Así, ADDQ 5,D2 sumará 5 al contenido de D2, y la instrucción entera ocupa una palabra.

Hay que hacer una puntualización importante sobre las instrucciones ADD que hemos examinado hasta aquí: y es que no incluyen arrastre alguno en la suma destino. Si queremos esto (en especial para operaciones aritméticas en doble precisión) deberemos emplear la instrucción ADDX. Así ADDX D2,D4 sumará los contenidos de D2 y D4, poniendo el bit de extensión en SR y almacenando el resultado en D4. Supongamos que D2 contenga 0000 0000 y D4 sea 0000 0001 con X=1; pues bien, tras la ejecución de ADDX tendremos en D4 0000 0010. Por el contrario, si hubiéramos empleado ADD, el contenido final de D4 sería 0000 0001.

El siguiente grupo de instrucciones aritméticas que es necesario examinar es el grupo SUB (restar, o *subtract*). Este conjunto es complementario al de las ADD, con las mismas restricciones en el direccionamiento y repercusiones sobre el código de condición. Los ejemplos que damos a continuación son todos válidos al tiempo que típicos:

```
SUB  D2,D3     Restar D2 de D3
               y poner el resultado en D3
SUBA #4,A3     Restar 4 de A3
SUBI #200,D2   Restar 2 de D2
SUBQ #1,D2     Resta rápida D2 menos 1
SUBX D2,D4     Da a D4 el resultado D4-D2-X
```

Es de notar que Motorola no proporciona una instrucción independiente de incremento o decremento. Para realizar estas funciones es preciso recurrir a las versiones rápidas de ADD y SUB (¡después de todo, sólo ocupan una palabra!).