Caer en una subrutina

Llegados a este punto, examinaremos la instrucción JSR ("Jump to subroutine": salto a la subrutina) y su utilidad práctica

Para mostrar cómo actúa esta instrucción, comencemos con un sencillo programa que transforma una tabla de 10 palabras de longitud mediante la fórmula:

```
TARRAY[I]:=ARRAY[I]^ 2+20*ARRAY[I]
```

y obtiene la suma de todos los elementos que componen TARRAY colocándola en la posición SUM. La fórmula parece un tanto complicada, pero lo único que expresa es "toma cada elemento de la tabla, elévalo al cuadrado, suma 20 veces dicho elemento, y después almacena el resultado en la casilla correspondiente de la nueva tabla, TARRAY".

La implementación del 68000 para esta transformación se muestra en el "listado uno". Para comprobar la corrección del programa debemos examinar los valores de TARRAY para los datos de comprobación (del 1 al 10) almacenados en ARRAY. Si usted ejecuta el programa, los valores almacenados serán, naturalmente, 21, 44, 69, 96, etc. Observe que el problema no especifica el intervalo de números que se han de usar en ARRAY. Si nuestras respuestas han de ser correctas, el resultado de la transformación (y la suma) se ajustarán a la longitud de una palabra, es decir, 16 bits.

Pero ésta no es, ni mucho menos, la única solución del problema, pues existen otros muchos caminos para llegar al mismo resultado. Esto ya depende del diseño de programa, y es en esta fase del proceso de codificación cuando cobra importancia el empleo de la subrutina, especialmente en el lenguaje assembly.

Hay muchos métodos para diseñar programas, pero una manera de alcanzar lo que pudiéramos llamar un programa "aseado" (al tiempo que resulta fácilmente comprensible cuando se vuelve a revisar) es el llamado *desglose funcional*. Se trata de un método que tiene por finalidad la elaboración de secciones de programas bien definidas que realicen ciertas funciones con los datos. Las funciones se describen mediante verbos tales como "calcular", "trasladar" o "comprobar", y estas secciones de código, en un lenguaje de alto nivel, se denominan *procedimientos*, con los datos facilitados a través de los *parámetros*.

Al nivel de la codificación en ensamblador, la única construcción modular adecuada disponible es la subrutina, que en el 68000 se llama por medio de:

```
JSR     SUBR
```

Esto altera el flujo del programa para realizar un salto a la sección de código etiquetada como SUBR. El código que está en SUBR se ejecutará hasta encontrar RTS, en cuyo punto el control del flujo del programa se devuelve a la instrucción siguiente a JSR (el mnemónico JSR significa *Jump to SubRoutine* [salto a la subrutina] y RTS quiere decir *ReTurn from Subroutine* [retorno de subrutina]).

Volvamos a nuestro programa ejemplo. Supongamos que en vez de realizar el cálculo de elevar al cuadrado, multiplicar y sumar "en una línea", deseamos "descomponer" esto con una llamada a la subrutina para conseguir un mejor diseño. Acabaremos con un programa algo así como el mostrado en el "listado dos". Todo lo que ha cambiado ha sido que en la línea 6 tenemos una llamada de subrutina a CALC, y lo que antes aparecía en las líneas de la 6 a la 9 se contiene ahora en las líneas 12 a la 15, con una RTS colocada al final.

La ventaja está en que ahora poseemos una sección de código bien definida (las líneas de la 12 a la 16), que realiza una operación específica sobre los datos pasados en D1 (un parámetro). No sólo es un buen diseño sino que podemos llamar a este código, CALC, ¡tantas veces como precisemos para que repita el mismo cálculo sobre otros datos siempre contenidos en D1 (y depositados en D2)!

Otro detalle a tener en cuenta cuando se implementan subrutinas, en especial en entornos de multiusuario, es que se tiene por una buena práctica el disponer de una sola entrada y una sola salida para las subrutinas. Esto quiere decir que debemos evitar las llamadas a otras subrutinas desde dentro de una subrutina. Tampoco es recomendable implementar un gran número de *returns* condicionados, que hagan la depuración una tarea difícil.

Ventajas de las subrutinas

La subrutina no sólo favorece el buen diseño de un programa, sino que proporciona programas económicos en lo que a ocupación de memoria se refiere. Esto se debe a que ahora no será necesario repetir el código de CALC cada vez que debamos emplearlo. Es adaptable tanto en el sentido de poderla llamar desde cualquier punto del programa como el que si deseamos introducir alguna modificación sólo tendremos que mirar un fragmento determinado del programa para realizarla.

Por ejemplo, supongamos que deseamos esta transformación:

```
D2:=D1^2-20*D1
```

la nueva subrutina CALC cambiará su línea 15 así:

```
SUB     D1,D2
```

y asunto concluido. Es más, el programa, además de ser adaptable, resulta fácil de depurar. Por

Listado uno

```
* Transformación de ARRAY en TARRAY mediante
* TARRAY[I]:=ARRAY[I]^2+20*ARRAY[I]
* Ambas tablas constan de 10 elementos
* Los registros empleados son:
*   A0 apunta a ARRAY
*   A1 apunta a TARRAY
*   D0 es un contador de bucle
*   D1 es un almacenamiento temporal
*   D2 es una copia de ARRAY[I]

          ORG     $1000
1 START   MOVEQ   #10,D0          * establece contador bucle
2         LEA     ARRAY,A0        * establece el primer puntero
3         LEA     TARRAY,A1       * y el segundo
4         CLR     SUM             * prueba si la suma es cero
5 LOOP    MOVE    (A0)+,D1        * toma ARRAY[I]
6         MOVE    D1,D2           * y la copia
7         MULS    D2,D2           * hace el cuadrado de ARRAY[I]
8         MULS    #20,D1          * D1 llega a ser 20 veces ARRAY[I]
9         ADD     D1,D2           * suma elementos
10        MOVE    D2,(A1)+        * almacena en TARRAY[I]
11        ADD     D2,SUM          * y guarda el total dinámico
12        SUBQ    #1,D0           * decrementa contador bucle
13        BNE     LOOP            * itera si no es el fin
14        TRAP    #0              * salida a monitor
15 ARRAY  DC.W    1,2,3,4,5,6,7,8,9,10
16 TARRAY DS.W    10
17 SUM    DS.W    1
18        END
```

Notas al programa:
Entre las líneas 1 y 4 se inicializan las variables empleadas por el programa; después, en la línea 5, cada elemento de ARRAY es cargado en D1 mediante el direccionamiento indirecto con posdecremento. En la línea 6 se hace una copia en D2, de manera que no tenemos que acceder a la tabla de nuevo. Las líneas 7 y 8 forman dos componentes de la transformación en D1 y D2, mediante MULS, y la línea 9 suma ambas. La línea 10 almacena el resultado en el elemento correspondiente de TARRAY, y la línea 11 suma el resultado en la suma dinámica, SUM. Finalmente, las líneas 12 y 13 son sentencias de control del bucle que permiten transformar sólo los 10 elementos (entre 5 y 12/13 se establece, en efecto, un bucle FOR con el contador inicializado en la línea 1)

Listado dos

```
          ORG     $1000
1 START   MOVEQ   #10,D0
          LEA     ARRAY,A0
          LEA     TARRAY,A1
          CLR     SUM
5 LOOP    MOVE    (A0)+,D1
6         JSR     CALC            * llamada a subrut. para calcular
7         MOVE    D2,(A1)+        * nuevo valor de la tabla
8         ADD     D2,SUM
9         SUBQ    #1,D0
10        BNE     LOOP
11        TRAP    #0
12 CALC   MOVE    D1,D2           * subrut. para calc. D1^2+20*D1
13        MULS    D2,D2           * y poner el valor en D2
14        MULS    #20,D1
15        ADD     D1,D2
16        RTS                     *retorno al programa que llamó
```

ejemplo, podemos probar la validez de CALC mediante todo tipo de valores que daremos a D1 y examinar los resultados en D2 sin mayores complicaciones con otros trozos del programa o con datos irrelevantes. Sólo debe haber una entrada a la subrutina (en la etiqueta de dirección de la subrutina) y sólo una salida (en la instrucción RTS). Así podemos afirmar que la estructuración de nuestros programas con subrutinas facilitan las pruebas y la depuración.

Parece, pues, que así se satisfacen los criterios de un "buen programa", por lo menos desde el punto de vista de la economía, adaptabilidad y fiabilidad. Pero el empleo de subrutinas comporta también ciertas desventajas. Para percatarnos de ellas (y examinar la transmisión de datos por medio de parámetros) es necesario echar un vistazo al mecanismo de la pila en el 68000.

El mecanismo de la pila

Cuando ejecutamos la instrucción JSR es preciso recordar la dirección de la instrucción siguiente a aquella que llama a la rutina (el enlace con la subrutina). La instrucción JSR hace que se coloque en la pila la dirección de esa instrucción siguiente (se necesitan cuatro bytes para la dirección de una palabra larga completa) y que el contador del programa (PC) se cargue con la dirección de la subrutina. Cuando se ha ejecutado RTS en la subrutina, el PC se cargará con dicha dirección que será sacada de la pila, y la ejecución continuará después de la llamada a la subrutina. Toda esta manipulación de la dirección se hace al margen del usuario, pues el 68000 se encarga de todo. Pero hay efectos colaterales que el programador debe conocer.

Ante todo, hay que cerciorarse de que el puntero de la pila, SP (o bien, A7 en el 68000), es activado correctamente, de lo contrario podemos sobreescribir el código o los datos, ¡o incluso violar las direcciones del hardware! El modo habitual de hacer esto es:

```
STACK   EQU   $1000        *pone la pila a 1000 hexa
BEGIN   LEA   STACK,SP     *inicializa el puntero
                           de la pila
```

y la pila irá pasando del 1000 al cero (porque cada *push* o envío a la pila es un predecremento).

En segundo lugar, ¡debemos estar seguros de que la pila no haya crecido demasiado! Debemos calcular la cantidad de pila que necesitamos usar, lo cual es difícil. Para programas relativamente sencillos con sencillas llamadas a subrutinas y empleos de la pila, es posible tener una respuesta exacta mediante el examen del código. Para programas *anidados* (es decir, con subrutinas que llaman a otras subrutinas) o *recursivos* (una subrutina se llama a sí misma), el problema de dar un tamaño a la pila puede ser casi irresoluble, y se han de emplear técnicas de ensayo y error.

En el próximo capítulo daremos ejemplos pormenorizados de llamadas a subrutinas y, en particular, nos ceñiremos a los métodos para pasar los datos a y desde la subrutina, empleando en ciertos casos algunas instrucciones únicas del 68000.