Una pila de ideas

Nuestro examen se centra en las relaciones entre la subrutina y la pila, como paso previo al estudio de los parámetros

Comenzaremos analizando un ejemplo que realiza dos llamadas a la subrutina CALC, ya presentada anteriormente.

```
                ....          *líneas de código
2000 4EB8       JSR CALC      *llamada a la
                               subrutina
                (...líneas de código...)
2100 4EB8       JSR CALC      *otra vez
2102 4000
                (...líneas de código...)
4000 3200   CALC MOVE D0,D7   *entrada subrutina
                (...líneas de código...)
4100 4E75       RTS           *retorno a llamada
```

Cada vez que se ejecuta la instrucción JSR, el 68000 coloca el contenido del contador de programa (la dirección de la instrucción siguiente a la instrucción JSR) en la pila y carga la dirección de la rutina especificada por JSR en el PC. La subrutina se ejecuta seguidamente y cuando encuentra RTS el 68000 saca la dirección de retorno (conocida como *dirección de enlace*) de la pila y la pone en el PC. Es de observar que la dirección de enlace (*linkage*) se guarda en formato de palabra larga completa, por lo que en el ejemplo, tras la primera ejecución de CALC, la pila y el PC contendrán los valores que se ilustran en el dibujo (página contigua).

Después de la ejecución de RTS, al final de la subrutina CALC, se restaura la dirección de enlace en el PC y la pila queda como se ilustra en la segunda parte del dibujo. Es de observar que la dirección de enlace continúa escrita en la pila, y será sobreescrita la siguiente vez que ésta sea usada.

Este mecanismo de enlace es obra del hardware del 68000. Pero su significado es que si anidamos subrutinas, el mecanismo de enlace también se cuidará automáticamente de esta situación.

Es claro que este método de apilar las direcciones de enlace automáticamente se cuida de las subrutinas anidadas mediante la ampliación de la pila. Supongamos que CALC se llama a sí misma (llamada recursiva). De nuevo el mecanismo de enlace se encargará de la situación apilando todas las direcciones de enlace una encima de la otra hasta que acaben las llamadas recursivas, o hasta que no se venga abajo el funcionamiento por culpa de alguna violación de dirección, es decir, ¡porque la pila ha crecido en exceso!

Volvamos ahora al problema de cómo se pasan los parámetros, tanto de entrada como de salida. En el capítulo anterior pasamos datos a la subrutina CALC mediante D1 y recuperamos datos de ella con D2. Esta sencilla solución del problema basta para pequeños programas, pero precisamos una solución más general para programas mayores, sobre todo pensando en que sólo se dispone de unos cuantos registros.

Un método utilizado para superar esta dificultad, en especial con compiladores, es el empleo de la pila como instrumento para pasar parámetros. Así, escribiríamos esto para pasar parámetros a CALC:

```
2000   MOVE PARAM1,-(SP)  *pone el primer
                           parámetro en la pila
2004   MOVE PARAM2,-(SP)  *y el segundo
2008   JSR CALC           *llamada a CALC
200A   .....
```

donde la pila contendría, al entrar en CALC:

```
                PARAM1
                PARAM2
                enlace ls
SP en CALC→enlace ms
```

Para acceder a PARAM1 tendremos que utilizar un desplazamiento en el SP de modo que salte por encima de la dirección de enlace:

```
ADDQ  #6,SP     *ajuste puntero
MOVE  (SP),D4   *toma el
                 parámetro
```

o, más sencillamente:

```
MOVE  6(SP),D4
```

Naturalmente, si alteramos el puntero de la pila, del modo que sea, debemos cerciorarnos de que antes de ejecutar la instrucción RTS apunte a la dirección de enlace, de otro modo pueden sobrevenir resultados desastrosos e impredecibles. Este principio es también válido cuando CALC devuelve parámetros mediante la pila, a menos que no sobreescriba uno o los dos parámetros de entrada, como es obvio.

No obstante, un método más sencillo será el empleo de una pila independiente para los parámetros, por ejemplo, la A6, dejando tranquilo al puntero de la pila (que se reserva para las funciones del hardware). En este caso, la llamada sería:

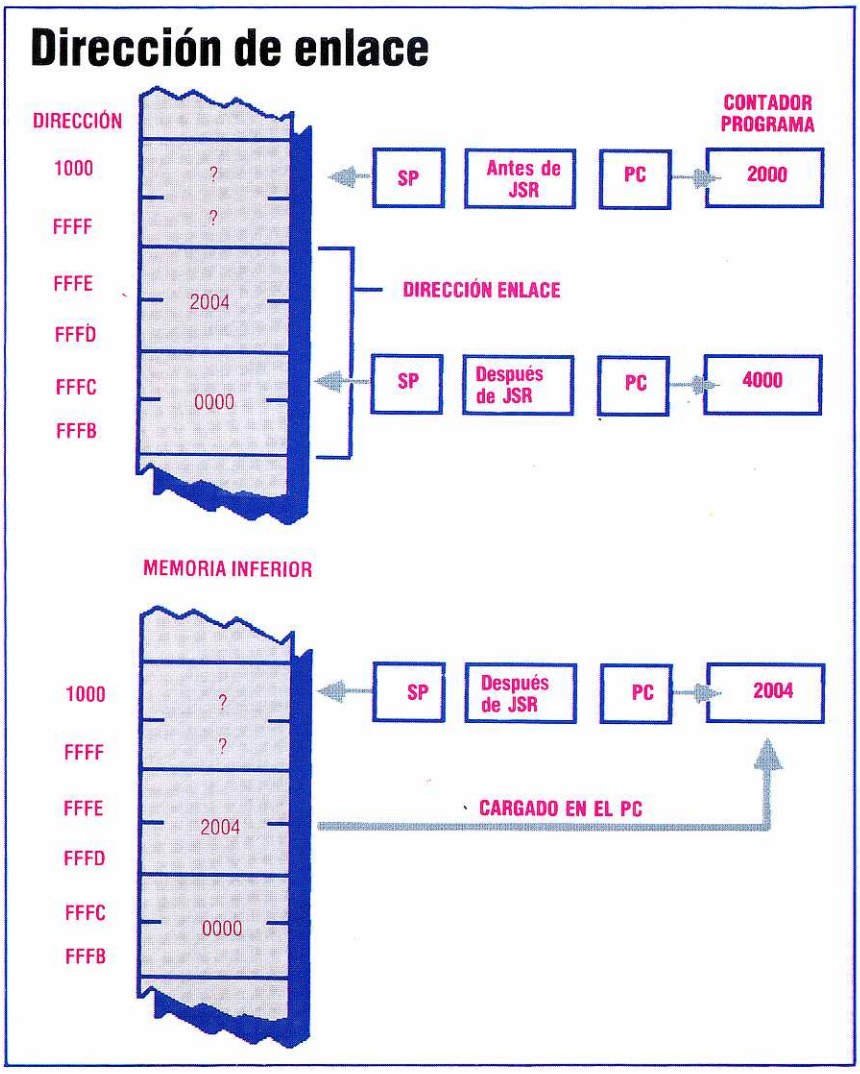
```
MOVE  PARAM1,-(A6)   *pone primero
                      el parámetro en la
                      pila A6
MOVE  PARAM2,-(A6)   *y el segundo
JSR   CALC           *guarda el SP
                      para el enlace
```

y dentro de la subrutina podemos fácilmente tomar los parámetros empleando, por ejemplo:

```
MOVE  (A6)+,D2       *toma el segundo
                      parámetro
MOVE  (A6)+,D1       *y el primero
```

ya que no existe el peligro de "pisar encima" de algún enlace de llamada a rutina. Es de notar que se toman los parámetros en el orden inverso respecto a la llamada.

Las pilas se emplean en los modernos lenguajes de alto nivel estructurados en bloques, como el PASCAL, en una forma muy ordenada y estructurada.



Vínculo permanente
El 68000 guarda una dirección de retorno (la *dirección de enlace*) en la pila antes de pasar el control del programa a la subrutina. Esta dirección se guarda en formato de palabra larga en la pila y se restaura en el contador de programa (*program counter*: PC) cuando se encuentra una instrucción RTS

La estructura de pila típica permite parámetros y variables locales de un procedimiento (que se implementará como subrutina en el código a ejecutar) que ha de permitir espacio en la pila dentro de un área definida. Así, por ejemplo:

```
Procedure Calcular (xval, yval:int);
var
        store:int;
begin
        store:=2*xval+yval^2;
```

como un fragmento de PASCAL tendría los parámetros de entrada xval e yval, y la variable local store, espacio reservado en la pila. Además, todo almacenamiento temporal sin nombre que necesite el compilador (pongamos por caso, para retener el componente yval^2 de store mientras se evalúa 2*xval) empleará la pila en su manera normal de poner y sacar.

Este arreglo está bien estructurado desde la perspectiva del compilador, pero en algunos ordenadores puede llevar a un nada despreciable dispendio en la manipulación de los datos de pila. En el 68000, la situación está facilitada considerablemente por el uso de dos instrucciones: LNK (*link*: enlazar) y UNLNK (*unlink*: desenlazar).

Estas instrucciones se emplean juntas y permiten la fácil manipulación de datos mediante la reserva de bloques de memoria dentro de la pila para uso de la subrutina. Después de una entrada a la subrutina, LNK establece un registro de direcciones definido (denominado *frame pointer*, FP, puntero marco) en un área de datos de la pila, y baja el SP de la pila un cierto número de posiciones. Por ejemplo, si la pila era:

```
        param1
        param2
        enlace ls
SP→     enlace ms
```

después de ejecutar JSR, el estado, tras la ejecución de la instrucción LNK, sería:

```
        param1
        param2
        enlace ls
        enlace ms
FP →    FP antiguo
|       denominado
desplazamiento local
|       variables
SP
```

El espacio de la pila crecería "hacia abajo", como indica el dibujo, dado que la subrutina necesita mayor espacio de trabajo.

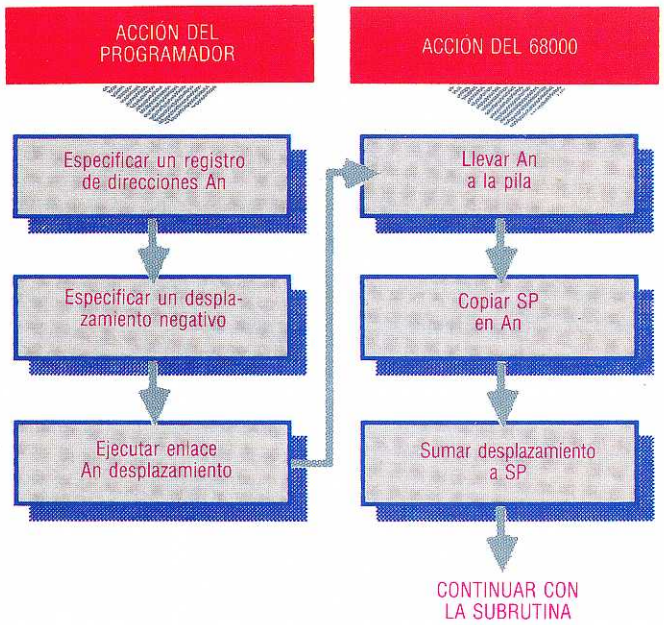
Una vez ejecutada la instrucción UNLNK al final de la subrutina, inmediatamente antes de la instrucción RTS, los punteros volverán a su estado previo a la entrada en la subrutina.

Hasta aquí hemos visto los dos extremos respecto del pase de parámetros en el 68000. En un extremo tenemos el empleo sencillo de los registros en el que el parámetro se carga en el registro de los datos (en realidad, un registro de direcciones si deseamos pasar una referencia a un parámetro). En el otro extremo tenemos el empleo altamente estructurado de la pila con el fin de implementar el pase de parámetros en un lenguaje de alto nivel. Veamos ahora un término medio, un estado en que podemos hacer un empleo algo más sofisticado que la utilización del registro sencillo pero que no implica el uso de la pila.

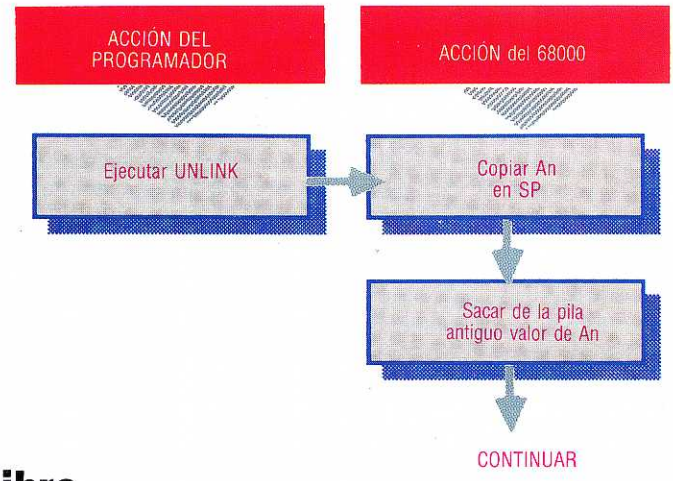
El primer método consiste en emplear un área definida de datos dentro de un grupo de subrutinas (p. ej., llamado módulo) que contiene todos los parámetros de entrada/salida asociados con ese módulo. Esta área de datos no es global, pero sólo es empleada por un conjunto definido de subrutinas para el propósito específico de pasar parámetros. Cada subrutina sabrá exactamente dónde tomar los parámetros y dónde dejar su parámetro de salida al final. Así, por ejemplo, podríamos escribir:

```
CALC    LEA     INPUTS,A4       *apunta al
                                 área parám. entrada
        MOVE    (A4),D0         *toma el primer
                                 parámetro
        MOVE    (A4)+,D1        *y también el
                                 segundo
        ....                    *instrucciones
                                 diversas
        MOVE    D5,OUTPUT       *entra el
                                 parám. salida
        RTS
```

Naturalmente, podemos pasar un puntero al área de parámetros en un registro de direcciones definido. Esto significaria que la primera línea en el ejemplo anterior resulta innecesaria. En efecto,

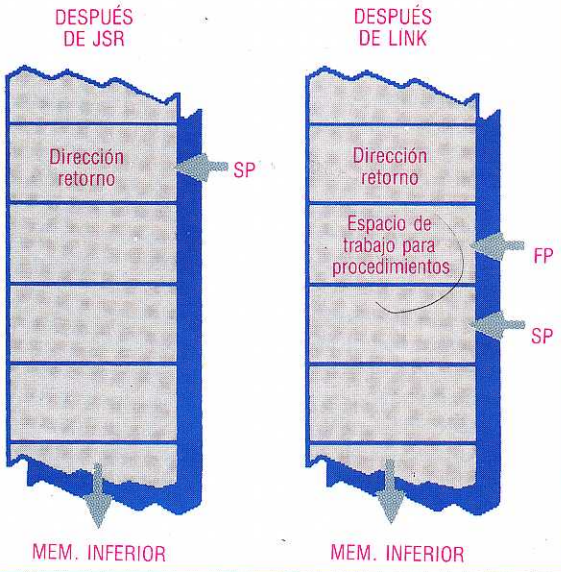


UNLINK



Pista libre

LINK y UNLINK proporcionan al programa unos medios rápidos de reserva de espacio de pila para la subrutina. Esto es importante en especial en un entorno de multiproceso, donde una subrutina puede ser interrumpida por otro programa. LINK toma un desplazamiento especificado por el usuario y hace descender el puntero de la pila en la memoria. Un registro, el An, se emplea como *puntero marco* por la rutina para acceder a los datos almacenados en el bloque reservado. Finalmente, SP y An pueden ser restaurados con sus valores previos mediante UNLINK. Este esquema muestra primero las acciones que se realizan y, en la parte inferior, el efecto en la pila



Caroline Clayton

este método se basa en la creación de áreas independientes de pilas para los parámetros de ENTRADAS y SALIDA.

El segundo método vuelve al procedimiento más sencillo de todos, el del uso directo de los registros, pero se aprovecha de la instrucción MOVEM. Examinemos primero esta instrucción. Sea la siguiente formulación:

```
MOVEM lista registro,AREARESERVA
```

Esto guardaría los registros definidos en la lista de registros en la dirección absoluta AREARESERVA y siguientes posiciones hacia arriba.

Por ejemplo:

```
MOVE D3/D5/A2, AREARESERVA
```

cargará D3 en AREARESERVA, D5 en AREARESERVA+2, y A2 en AREARESERVA+4. Podríamos guardar, asimismo, los registros en la pila con:

```
MOVEM D3/D5/A2,–(SP)
```

que apilará el registro en direcciones consecutivas decrecientes. Si los registros consecutivos se han de guardar, entonces el ensamblador acepta la versión taquigráfica:

```
MOVEM D1–D5/A4,–(SP)
```

donde se apilarán desde D1 hasta D5 y A4.

Podemos también almacenar los registros empleando la lista de registros como destino en la instrucción MOVEM. Por ejemplo:

```
MOVEM–(SP),D1–D5/A4
```

restaurará los registros almacenados en el ejemplo anterior.

Debemos preguntarnos para qué nos sirve esto en el pase de parámetros. La respuesta está en la conveniencia de almacenar registros que se han distribuido para un objetivo específico, por ejemplo para uso del sistema y después quedar libres para usar los registros en lo que queramos.

Por ejemplo, si reservamos D0 y D1 como registros de parámetros, entonces con tal de que almacenemos los restantes registros en la entrada a la subrutina, podríamos utilizar los restantes registros según se exija en la subrutina.

Por ejemplo:

```
CALC    MOVEM D2–D7,–(SP)   *guarda antiguos
                             D2-D7
        MOVE   D0,D2        *y carga el
                             primer parámetro
        ....                *instrucciones
                             que emplean
                             D2-D7
        MOVEM –(SP),D2–D7   *y restaura
                             valores antiguos
        RTS
```

De esto se puede colegir que el 68000 nos permite el empleo de varios métodos para pasar parámetros a las subrutinas y ¡nos proporciona además algunas herramientas útiles para hacerlo!