Apéndice A

Gestión de la memoria en tiempo de ejecución

A1. Organización de la memoria en tiempo de ejecución

Las técnicas de gestión de la memoria durante la ejecución del programa difieren de unos lenguajes de programación a otros, e incluso de unos compiladores a otros.

Para datos, básicamente la memoria se divide en:

- Memoria Estática.
- La Pila.
- El Monton (Heap).

A.1.1 Memoria Estática

La forma más sencilla de almacenar el contenido de una variable en memoria, en tiempo de ejecución, es hacerlo en la memoria estática. Así, el almacenamiento de dichas variables será permanente (durante la ejecución del programa).

Por ello, resulta obvio que los datos etiquetados como constantes y las variables globales de un programa tengan asignada la memoria necesaria durante toda la ejecución del programa.

Sin embargo, no todas las variables pueden almacenarse estáticamente. Para que una variable pueda ser almacenada en memoria estática, es necesario conocer su tamaño (número de bytes necesarios para su almacenamiento) en tiempo de compilación. Como consecuencia, aunque una variable (u objeto) sea de ámbito global, no podrán ocupar almacenamiento estático:

- Los objetos que correspondan a procedimientos o funciones recursivas, ya que en tiempo de compilación no se conoce el número de variables que serán necesarias.

- Las estructuras dinámicas de datos tales como listas, árboles, etc. ya que el número de elementos que la forman no es conocido hasta que el programa se ejecuta.

Las técnicas de asignación de memoria estática que utilizan los compiladores, son sencillas. A partir de una posición señalada por un puntero de referencia se aloja la variable X, avanzando el puntero tantos bytes como sean necesarios para almacenarla. Desde ese momento, el compilador sustituirá cualquier referencia a dicha variable por la de su dirección asignada. Subsiguientes constantes o variables globales serán acomodadas en memoria haciendo avanzar, de la misma manera, el puntero de referencia, hasta que el compilador les asigne espacio de memoria a todas ellas.

La asignación de direcciones de memoria se hace en tiempo de compilación y las variables globales estarán vigentes desde que comienza la ejecución del programa hasta que termina.

A.1.2 La pila

La aparición de lenguajes con estructura de bloque trajo consigo la necesidad de técnicas de alojamiento en memoria más flexibles, que pudieran adaptarse a las demandas de memoria durante la ejecución del programa.

Para entender esta necesidad, tomemos como ejemplo el lenguaje C, en el cual las funciones pueden tener variables locales, cuyo ámbito de visibilidad se restringe al código de su función y que tienen un tiempo de vida que se ajusta al tiempo de ejecución de la función: obviamente, sería innecesario que dichas variables ocupen memoria de forma estática en todo momento. Además, si así fuera y el compilador optara por asignar memoria estática a las variables locales de una función, sería imposible la utilización de técnicas de programación como, por ejemplo, la recursividad.

En los compiladores, en general, la asignación de memoria de variables locales se hace de una forma flexible, atendiendo al hecho de que solamente necesitan memoria asignada desde el momento que comienza la ejecución de la función hasta el momento en que ésta finaliza. Así, cada vez que comienza la ejecución de un procedimiento (o función) se crea un registro de activación para contener los objetos necesarios para su ejecución, eliminándolo una vez terminada ésta.

Dado que durante la ejecución un programa es habitual que unos procedimientos llamen a otros y estos a otros, sucesivamente, se crea cadena jerárquica de llamadas a procedimiento. Dado que la cadena de llamadas está organizada jerárquicamente, los distintos registros de activación asociados a cada procedimiento (o función) se colocarán en una pila en la que entrarán cuando comience la ejecución del procedimiento y saldrán al terminar el mismo.

La figura A.1b muestra cómo un compilador emplaza las variables estáticas en memoria, quedando la pila reservada para emplazar los registros de activación de los procedimientos en ejecución (los registros de activación se apilan en el mismo orden jerárquico de la cadena de llamadas). La estructura de los registros de activación varía de unos lenguajes a otros, e incluso de unos compiladores a otros.

En general, los registros de activación de los procedimientos suelen tener algunos de los campos que pueden se representan en la figura A.1a.

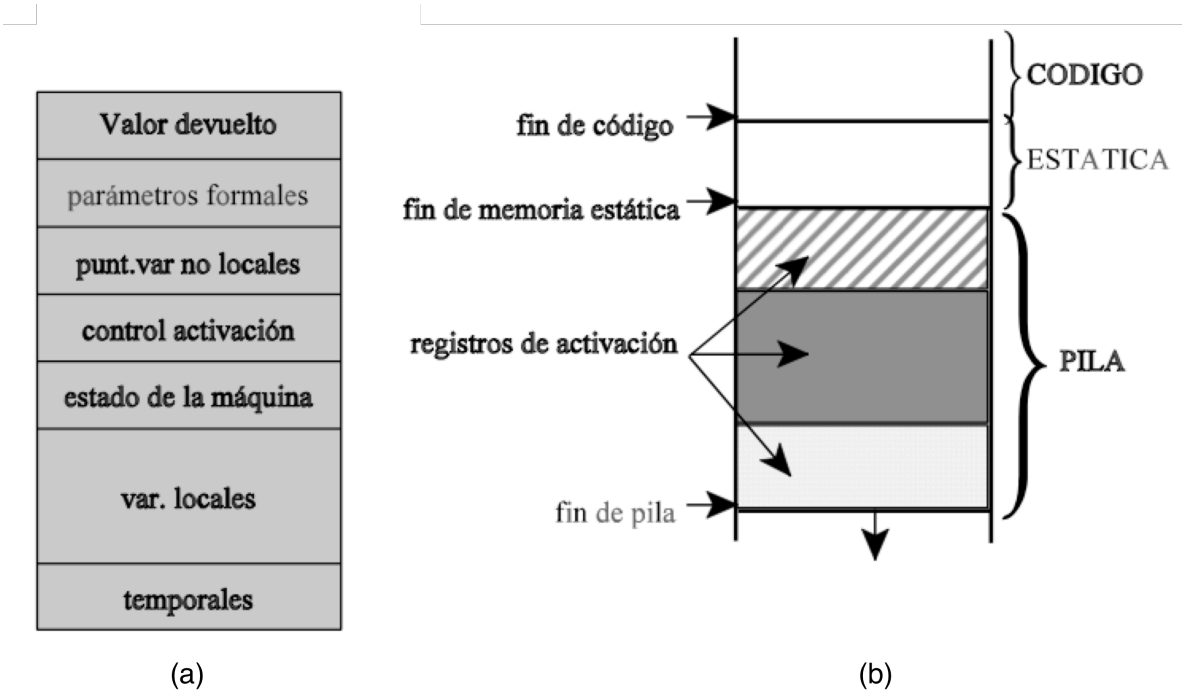

Figura A.1: a) Estructura de registro de activación. b) Estructura de la pila.

Como se ha mencionado, esta es una estructura muy general y algunos compiladores pueden carecer de algunos campos. De todas formas, a continuación se describen un poco más detenidamente cada uno de dichos campos:

- El puntero de control de activación guarda el valor que tenía el puntero de la cima de la pila (TOS, Top Of Stack) antes de que entrase en ella el nuevo registro, de esta forma una vez que se desee desalojarlo, puede restituirse el puntero de la pila a su posición original. Es decir, es el puntero que se usa para la implementación de la estructura de datos "Pila" del compilador.

  Como puede verse, el proceso de eliminar un registro de activación de la pila consiste, simplemente, en ignorarlo restaurando el anterior TOS (esto se verá mejor posteriormente en el ejercicio al final del apéndice).

- En la zona correspondiente al estado de la máquina se almacena el contenido que hubiera en los registros de la máquina antes de comenzar la ejecución del procedimiento. Obviamente, estos valores se almacenan porque necesitaran ser repuestos al finalizar la ejecución del procedimiento. El código encargado de realizar la copia del estado de la máquina es común para todos los procedimientos.

- El puntero a las variables no locales permite el acceso a las variables declaradas en otros procedimientos. Normalmente no es necesario usar este campo puesto que puede conseguirse lo mismo con el puntero de control de activación. Este campo sólo tiene sentido cuando se utilizan procedimientos recursivos.

- La zona correspondiente a los parámetros formales (variables que aparecen en la cabecera del procedimiento) alojará los parámetros pasados al procedimiento. Por ejemplo, supongamos que la función 'q' está definida en C como:

```
int q(char a, char b)
{
     int c, d;

     c=2*a;
     b=3*b;

     return (c+d);
}
```

  en este caso la zona de parámetros será de 2 bytes, ya que hay dos parámetros (a y b), y ambos ocupan 1 byte dado que son de tipo *char*.

- La zona correspondiente a las variables locales (las que se definen dentro del bloque o procedimiento) asigna memoria para alojar a las variables locales al procedimiento. Por ejemplo, si tomamos la función q( ) definida anteriormente, vemos que hay dos variables locales, ambas de tipo *int*. Suponiendo que las variables *int* son de 16 bits para el compilador que estemos utilizando, entonces el compilador dimensiona la zona de variables locales con 4 bytes: 2 bytes para la variable c (pues es tipo *int*) , y otros 2 bytes para la variable d, que también es tipo *int*.

- La zona de valor devuelto es aquella mediante la cual se retorna el valor devuelto por la función. En el ejemplo que nos ocupa será de 2 bytes, pues el valor devuelto está definido con el tipo *int*.

- La zona de valores temporales la dimensiona el compilador para utilizarla para el cálculo de expresiones. Por ejemplo, si realizamos la llamada q((3*5)+(2*2),7) puede ser que el compilador necesite alguna variable temporal para el cálculo de (3*5)+(2*2), para lo cual asignará el espacio necesario a la zona de valores temporales y echará mano de ella como almacenamiento auxiliar para la realización del cálculo (en algún sitio debe haber una zona de memoria que contenga el valor intermedio 15, y el valor intermedio 4 para sumarlos a continuación).

Cabe decir que para dos procedimientos diferentes los registros de activación no tienen por qué tener el mismo tamaño. Pueden tener tamaños diferentes. Este tamaño por lo general es calculado en tiempo de compilación ya que el compilador dispone de información suficiente sobre el tamaño de los objetos que lo componen. Aunque en ciertos casos esto no es así, como por ejemplo ocurre en C cuando se utilizan arrays de dimensión indefinida. En estos casos el

registro de activación debe incluir una zona de desbordamiento al final, cuyo tamaño no se fija en tiempo de compilación sino solo cuando realmente llega a ejecutarse el procedimiento. Esto complica un poco la gestión de la memoria, por lo que algunos compiladores de bajo coste suprimen esta facilidad.

Si bien ya conocemos la estructura de un registro de activación y cómo estos registros se apilan cuando se realiza una llamada u ocurre un retorno, vamos a ver con más detenimiento qué ocurre cuando un procedimiento p llama a otro procedimiento q.

El procedimiento de gestión de la pila, cuando un procedimiento p llama a otro procedimiento q, se desarrolla en dos fases, la primera de ellas corresponde a las acciones del código que incluye el procedimiento p antes de transferir el control a q, y la segunda, al código que debe incluirse al principio de q para que se ejecute cuando reciba el control.

- Primera fase (se realiza en la función llamante, antes de saltar a la función llamada):

  - El procedimiento que realiza la llamada (p) evalúa las expresiones de la llamada, utilizando para ello su zona de variables temporales. Por ejemplo, en caso que desde el procedimiento p se realice la llamada q((3*5)+(2*2),7) , el cálculo de (3*5)+(2*2) se realiza utilizando la zona de variables temporales de p.

  - El resultado de las expresiones de la llamada se copia en la zona de parámetros formales del procedimiento que recibe la llamada (q).

  - El procedimiento que realiza la llamada (p) coloca el puntero de control del procedimiento al que llama(q) de forma que apunte al actual final de la pila (TOS), tras lo cual y transfiere el control al procedimiento al que llama (q).

- Segunda fase (se realiza en la función llamada):

  - El procedimiento llamado (q) salva el estado de la máquina antes de comenzar su ejecución, usando para ello la zona correspondiente de su registro de activación.

  - El receptor de la llamada (q) inicializa sus variables y comienza su ejecución.

  - ..... (ejecución de q) .....

Al terminar la ejecución del procedimiento llamado (q), se desaloja su registro de activación procediendo también en dos fases:

- Primera fase (se implementa mediante instrucciones al final del procedimiento que acaba de terminar su ejecución)

  - El procedimiento saliente (q) antes de finalizar su ejecución coloca el valor de retorno al principio de su registro de activación.

```
funcion p ( ... // parámetros formales p // ... )
{
      // var. locales de p
      .....
      x = q ( (3*5) + (2*2), 7 );
}

función q ( ... // parámetros formales q // ... )
{
      // var. locales de q
      .....
      .....
      .....
}
```

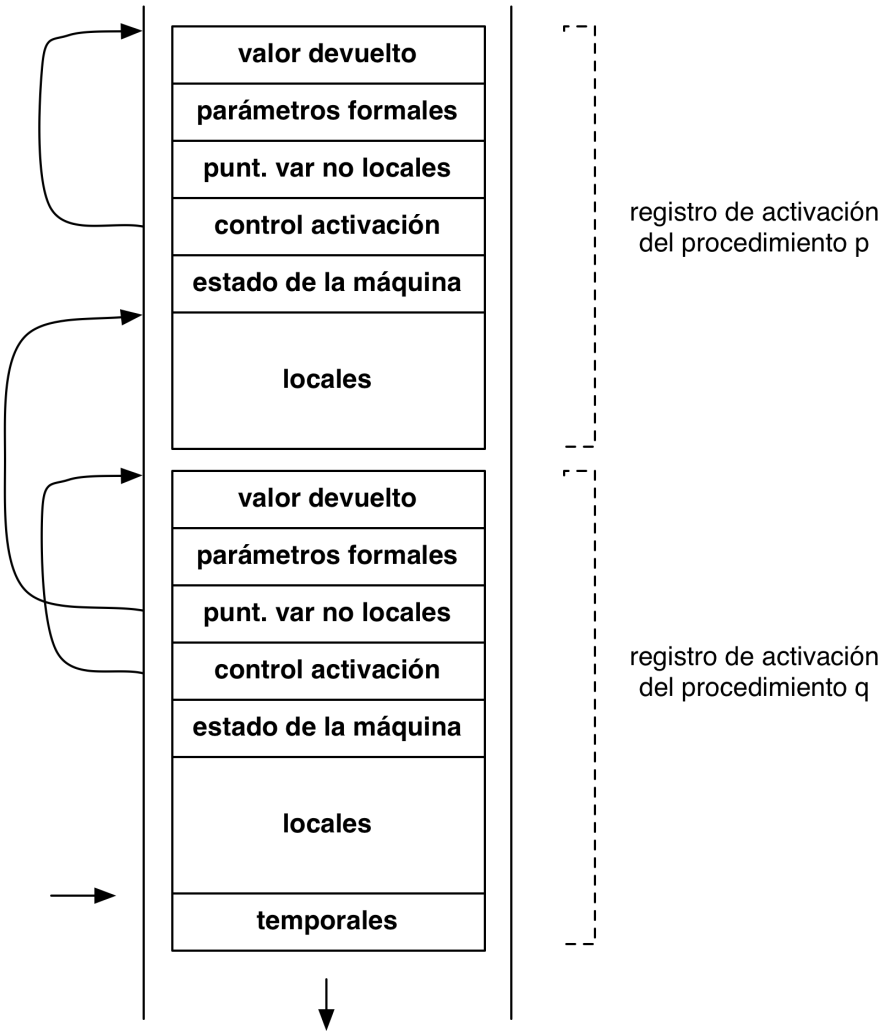

Figura A.2: Registros de activación en pila durante la ejecución del procedimiento q.

- Usando la información contenida en su registro el procedimiento que finaliza (q) restaura el estado de la máquina y coloca el puntero de final de pila (TOS) en la posición en la que estaba originalmente.

- Segunda fase (en procedimiento que hizo la llamada, se realiza tras recobrar el control):

  - El procedimiento que realizó la llamada (p) copia el valor devuelto por el procedimiento llamado (q) dentro de su propio registro de activación (de p).

La figura A.2 muestra el estado de la figura (durante el tiempo de ejecución cuando se alcanza la ejecución de las sentencias del procedimiento q, en el código de la figura A.2).

Dentro de un procedimiento, el compilador referencia siempre las variables locales, temporales o parámetros, utilizando direccionamiento relativo. Estas variables, locales al procedimiento, se referencian siempre como direcciones relativas al comienzo del registro de activación, o bien al comienzo de la zona de variables locales.

A.1.3 El Montón (Heap)

Cuando el tamaño de un objeto a colocar en memoria puede variar en tiempo de ejecución, no es posible ubicarlo en memoria estática, ni siquiera en la pila. Ejemplos de este tipo de objetos son: las listas, los árboles, etc. Para manejar este tipo de objetos el compilador debe disponer de un área de memoria de tamaño variable, que no se vea afectada por la activación o desactivación de procedimientos. Esta área de memoria se llama montón (traducción literal del termino ingles heap que habitualmente se utiliza en la literatura técnica).

Sin embargo, la descripción del montón excede el objetivo del presente curso.

A.2 Caso del compilador para TMS320C2x/2xx/5x

Hasta ahora se ha estudiado un registro de activación genérico. Si bien todos los compiladores utilizan registros de activación, no todos ellos son tan complejos. Compiladores sencillos como el de los DSP TI TMS320C2x/2xx/5x prescinden de varios de los campos vistos anteriormente, y dado que estos DSP se estudian en esta asignatura, vamos a tomar el caso concreto de este compilador de lenguaje C, para demostrar cómo funciona la gestión de las variables en pila.

A.2.1 Gestión de memoria para variables estáticas y globales

Se asigna un espacio continuo para cada variable global o estática declarada en un programa C.

A.2.2 Puntero a Pila, Puntero a Frame, Puntero a Variables Locales

El compilador gestiona una pila software, no se aprovecha la pila implementada en hardware en estos procesadores, dado que un nivel de profundidad 8 es insuficiente. De hecho la profundidad de la pila puede cambiarse mediante el enlazador (linker) utilizando el símbolo __STACK_SIZE. El tamaño por defecto de la pila es de 1K. Nótese la diferencia, razón por la cual no puede utilizarse la pila hardware.

El registro de activación (local frame) contiene: la dirección de retorno, zona de variables locales y zona de parámetros. El registro de activación es apilado para la ejecución de una función llamada, y desapilado cuando esta finaliza y retorna el control a la función que hizo la llamada.

Por lo tanto hay tres registros, el puntero a pila (Stack Pointer, SP), el puntero al registro (Frame Pointer, FP), y el puntero a variables locales (Local Variable Pointer, LVP), gestionados en la pila:

- El puntero a pila, SP, reside en el registro AR1. Este puntero siempre apunta a la siguiente palabra disponible en la pila.

  El puntero al registro de activación (también llamado frame), FP, reside en el registro AR0. Este puntero apunta al comienzo del actual frame (registro de activación).

- La primera palabra del frame, a la que apunta directamente FP, se usa como localización temporal de memoria para permitir transferencias registro a registro y es esencial para crear funciones C reentrantes.

- El puntero a la zona de variables locales, LVP, reside en el registro AR2. Todas los objetos almacenados en este registro de activación son accedidos utilizando este puntero mediante direccionamiento relativo.

A.2.3 Estructura de una función y convenciones de llamada

El compilador de C impone normas estrictas sobre las llamadas de funciones. Cualquier función que llama o es llamada desde C debe seguir estas normas.

La figura A.3 ilustra una llamada de función típica. En este ejemplo se pasan parámetros a la función y se utilizan variables locales. Este ejemplo también muestra la gestión de un frame local (registro de activación) para la función llamada.

A las funciones que no tienen argumentos ni variables local no se les asigna un frame local.

Cabe remarcar que frame y registro de activación son conceptos diferentes.

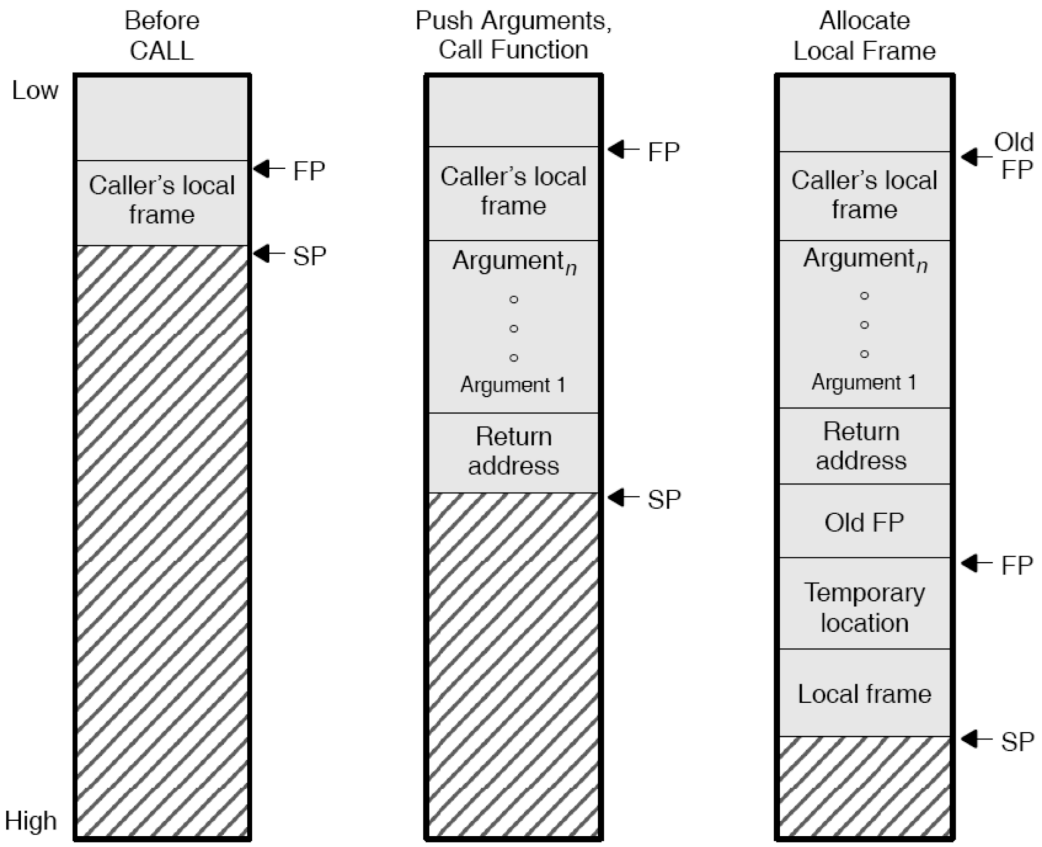

Figura A.3: Uso de la pila durante la llamada a una función.

A.2.4 Como funciona una llamada a función

La función que realiza una llamada realiza las siguientes tareas:

1. La función llamante apila los argumentos en pila (en orden inverso, el declarado en último lugar es el primero en apilarse).

2. La función llamante llama a la función llamada.

.... (se ejecuta la función llamada)

3. Cuando la función llamada se completa desapila los argumentos que apiló.

La función llamada debe realizar las siguientes tareas:

1. Apilar la dirección de retorno.

2. Apila el puntero a frame de la función llamante (Old FP).

3. Gestiona el frame local.

4. Si la función llamada modifica los registros AR6 o AR7 los apila.

..... (se ejecuta la función)

5. Si la función devuelve un escalar lo pone en el acumulador, si no cabe en el acumulador devuelve en el acumulador un puntero a la variable que contiene el valor devuelto.

6. Restaura AR6 y/o AR7 si fueron apilados.

7. Desapila el frame local.

8. Restaura el FP.

9. Copia la dirección de retorno de la pila software a la hardware.

10. Retorna.

A.3 Ejemplo de gestión de variables en pila

En la figura A.4 se muestra un código fuente en lenguaje C, donde se indican varios puntos, en los cuales se pretende monitorizar el contenido de la pila y las variables globales, con el objeto de comprobar la gestión de descriptores y variables locales en la pila.

La figura A.5 muestra los contenidos de la pila en cada uno de los puntos indicados sobre el código fuente.

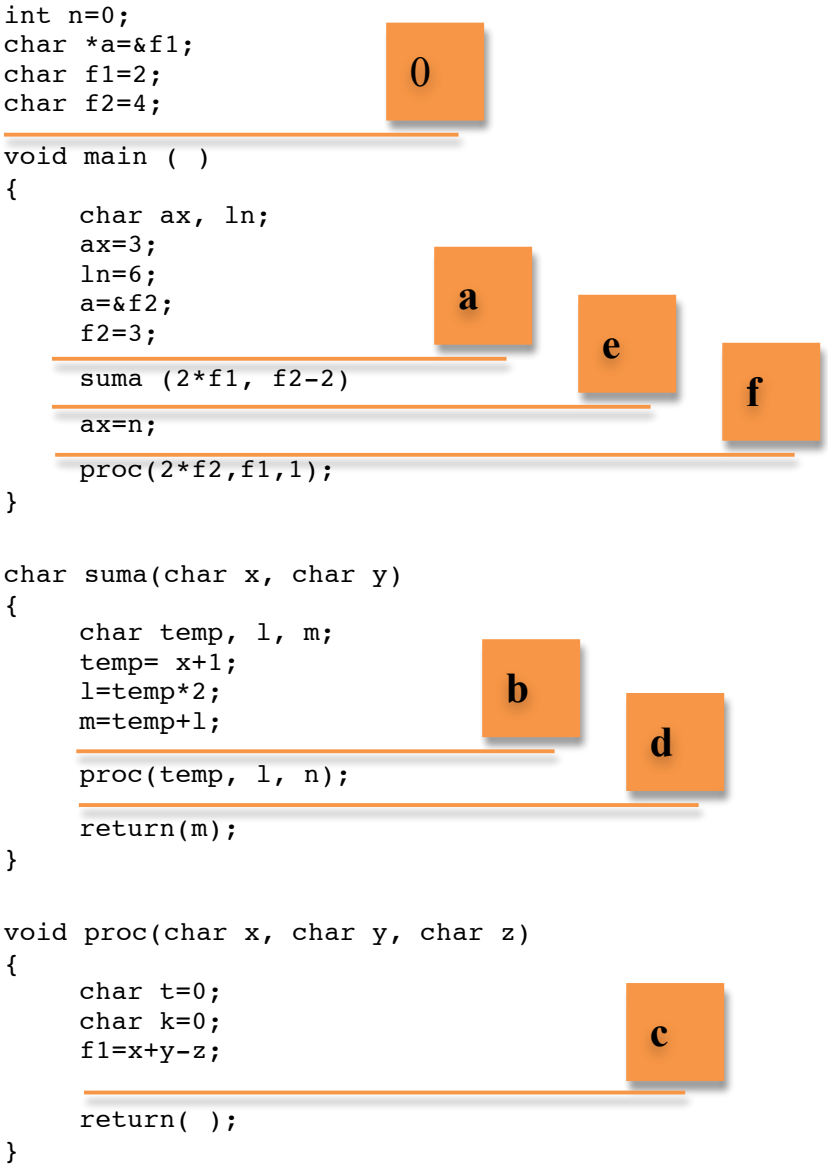

Figura A.4: Código fuente y puntos marcados para inspección de las variables en pila.

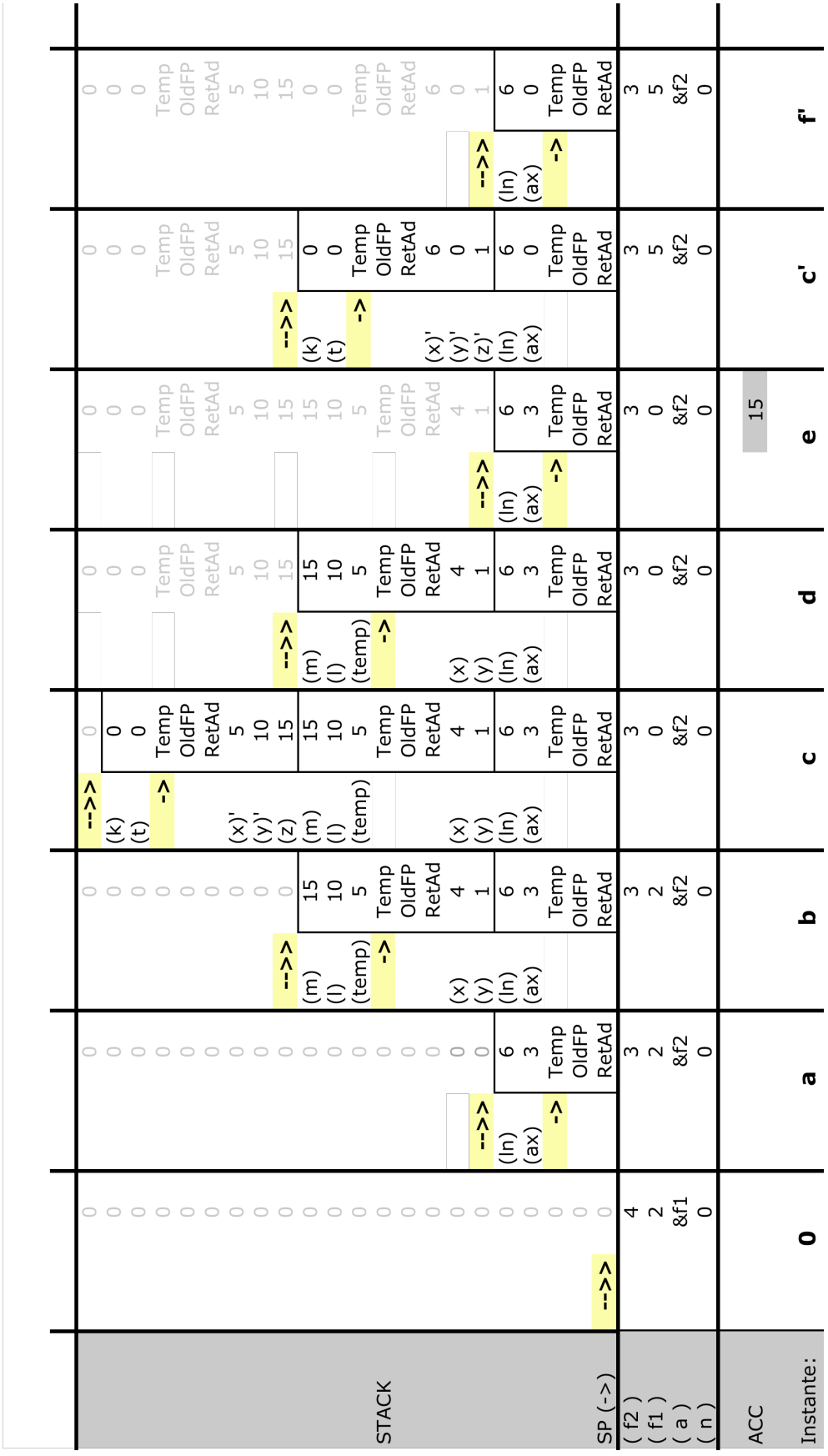

Figura A.5: Gestión de descriptores y variables en pila, en los puntos marcados sobre el código fuente de la figura anterior.