Procesadores de lenguajes. Práctica 11.

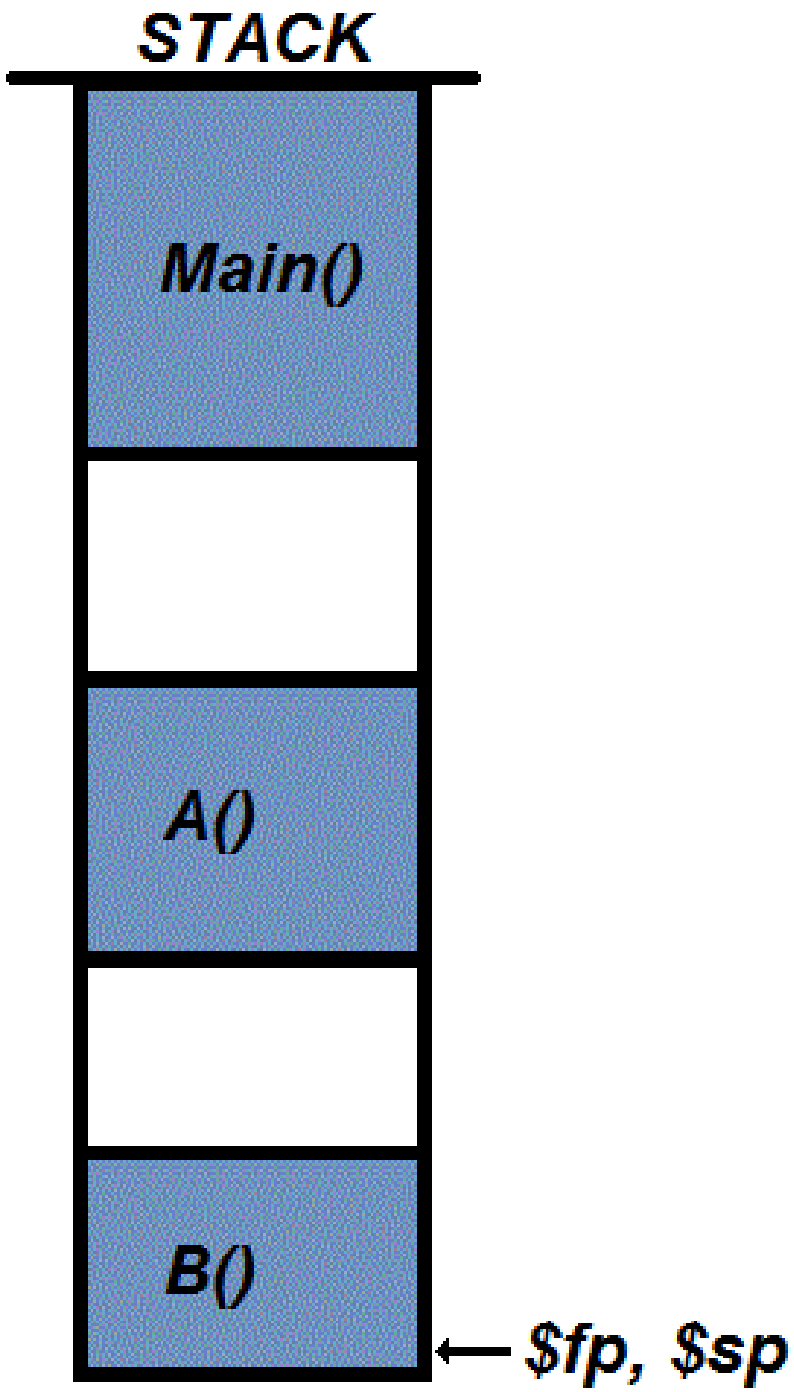
LA GESTIÓN DE LA MEMORIA EN EL COMPILADOR DE TINTO

Índice

1.1 La memoria de la pila

- El lenguaje Tinto es un lenguaje de programación muy sencillo, tanto en términos de tipos de datos como en las instrucciones incluidas en el lenguaje. Con respecto al uso de la memoria en tiempo de ejecución, los programas en Tinto no utilizan memoria dinámica (montículo). Por tanto, en tiempo de ejecución la gestión de la memoria se limita a la memoria de pila. Esta memoria de pila se dedica a almacenar los registros de activación de las funciones. La gestión de este tipo de memoria consiste en desarrollar los procesos de llamada a una función (que se traduce en apilar un nuevo registro de activación en la pila) y de retorno de una función (que se traduce en desapilar un registro de activación).

- El compilador de Tinto gestiona la memoria de pila en forma de pila decreciente, es decir, la base de la pila se encuentra en una posición de memoria alta y el crecimiento de la pila se dirige hacia zonas de memoria más baja. La siguiente figura muestra un ejemplo del estado de la pila en un instante de ejecución de un programa:



1.1 La memoria de la pila

La imagen anterior muestra el estado de la pila en un instante en el que la función Main() ha llamado en su ejecución a la función A() y ésta, a su vez, ha llamado a la función B(). El espacio entre los registros de activación de cada función está ocupado por los valores de los parámetros de llamada de la función. Es decir, el espacio entre el registro de la función A() y el de la función B() está ocupado por los parámetros de la llamada a la función B().

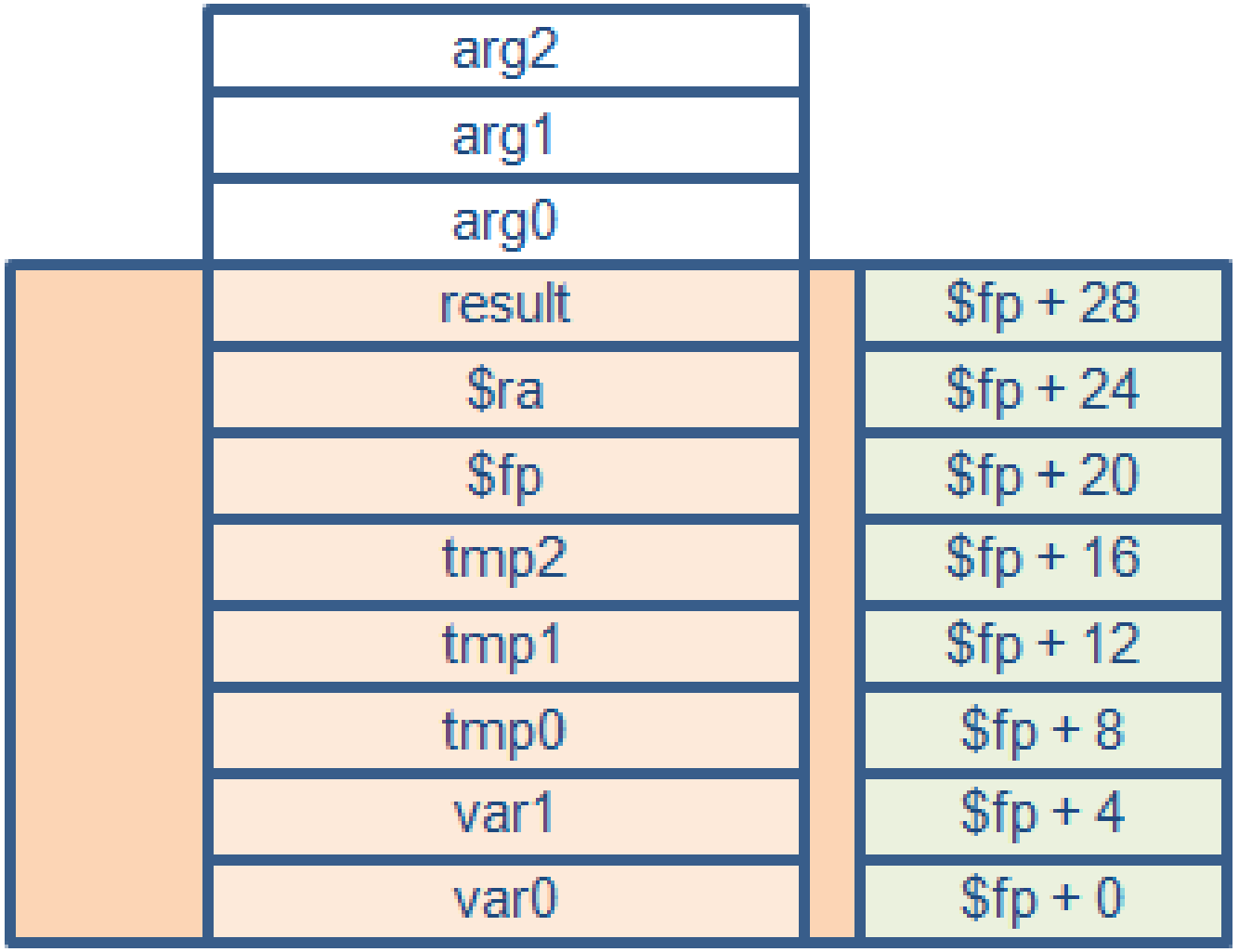
1.1 La memoria de la pila

- La gestión de la pila se desarrolla por medio de dos punteros, el puntero de registro y el puntero de pila:
  - El puntero de registro o frame pointer: Contiene la dirección base del registro de activación de la función, es decir, la dirección de memoria más baja del registro. Para acceder al contenido del registro se utilizan desplazamientos positivos respecto de esta dirección base. El valor de este puntero se almacena en un registro de propósito general de MIPS. Por convenio se utiliza el registro $30 para almacenar este valor. Normalmente en ensamblador se denota $fp, que en realidad no es más que el alias del registro $30.
  - El puntero de pila o stack pointer: Contiene la dirección de memoria de la cima de la pila, que en realidad es su valor más bajo ya que la pila crece "hacia abajo". Habitualmente su valor coincide con el frame pointer de la función que se está ejecutando, pero antes de la llamada a una nueva función se modifica su valor para reservar el espacio dedicado a los parámetros de la llamada a esa función. Su valor se almacena en un registro de propósito general de MIPS, que por convenio es el $29. Normalmente en ensamblador se denota como $sp, que se define como alias del registro $29.

1.2 El registro de activación de las funciones de tinto

- El registro de activación de una función almacena el valor de las variables temporales y locales de la función, así como información del estado del procesador antes de la ejecución de la función. Esto último permite restaurar el estado del procesador al término de la ejecución de la función.

- A continuación se muestra un ejemplo del contenido del registro de activación de una función, tal y como lo gestiona el compilador de Tinto. El ejemplo se refiere a una función con dos variables locales (var0 y var1), tres variables temporales (tmp0, tmp1 y tmp2) y tres argumentos (arg0, arg1 y arg2).



1.2 El registro de activación de las funciones de tinto

El frame pointer contiene la dirección de la posición más baja del registro. El compilador de Tinto coloca en primer lugar a las variables locales. Teniendo en cuenta que todos los tipos de datos de Tinto ocupan 4 bytes, las posiciones de memoria de cada variable se obtienen incrementando el puntero en bloques de 4 bytes. Por tanto, en este caso la variable local var0 está situada en la posición ($fp+0), mientras que la variable local var1 se encuentra en la posición ($fp+4).

1.2 El registro de activación de las funciones de tinto

- A continuación de las variables locales se colocan las variables temporales de la función. En este caso se trata de las variables tmp0, tmp1 y tmp2, que se colocan en las posiciones ($fp+8), ($fp+12) y ($fp+16).
- Por encima de las variables temporales se sitúa la información del estado del procesador. El compilador de Tinto no almacena el estado completo del procesador (que sería el valor de todos sus registros de propósito general y de la unidad de coma flotante) ya que no es necesario y además consumiría mucho tiempo en los cambios de función. En su lugar se almacenan sólo los valores de los registros $fp (que contiene la posición base del registro de activación de la función que realiza la llamada) y $ra (que contiene la dirección de retorno de la función, es decir, la posición de la instrucción de la función llamante que se debe ejecutar al término de la función llamada).
- En la parte más alta del registro de activación se reserva el espacio para almacenar el valor que debe devolver la función. Cuando termina la ejecución de la función y se desapila su registro de activación, el stack pointer se encuentra situado en la posición del primer argumento (arg0), de manera que la función llamante puede obtener el resultado de la función llamada accediendo a la posición ($sp-4).

1.2 El registro de activación de las funciones de tinto

- Los argumentos de llamada de la función no forman parte de su registro de activación sino que se encuentran encima de él. Como el compilador de Tinto conoce el tamaño del registro de activación (32 bytes en el ejemplo de la figura), puede acceder a estos argumentos mediante desplazamientos respecto al frame pointer. Por ejemplo, el primer argumento (arg0) se encuentra en la posición ($fp+32), el segundo argumento (arg1) se encuentra en la posición ($fp+36) y el tercer argumento (arg2) se encuentra en la posición ($fp+40).

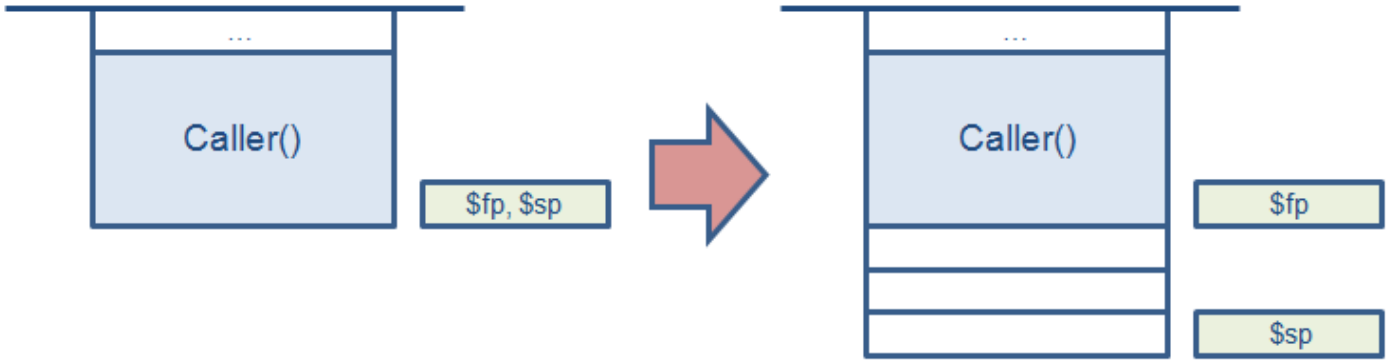
1.3 El proceso de llamada a una función

- El proceso de llamada de una función se produce cuando la ejecución de una función (que se denomina llamadora o caller) contiene la llamada a otra función (que se denomina destinataria o callee). Este proceso provoca que se coloque un nuevo registro de activación en la pila (el de la función destinataria) y que se produzca un cambio de contexto de ejecución, pasando a ejecutarse el código de la función destinataria.

- El proceso de llamada que desarrolla el compilador de Tinto se compone de los siguientes pasos:

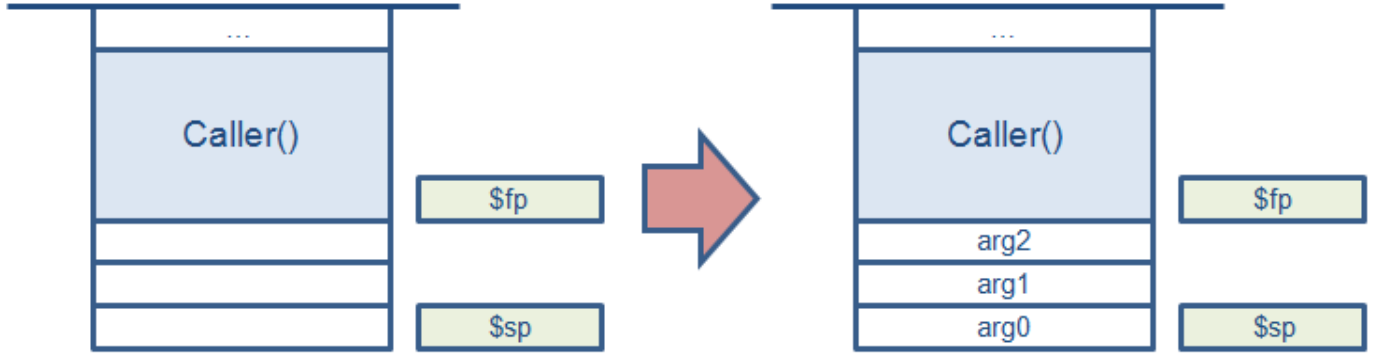
1.3 El proceso de llamada a una función

En primer lugar se reserva espacio en la pila para almacenar los valores de los argumentos de la llamada. Esto consiste en desplazar el *stack pointer* hacia abajo, abriendo un hueco por debajo del registro de activación de la función llamadora. Esta acción corresponde a la traducción de la instrucción de código intermedio PRECALL.



1.3 El proceso de llamada a una función

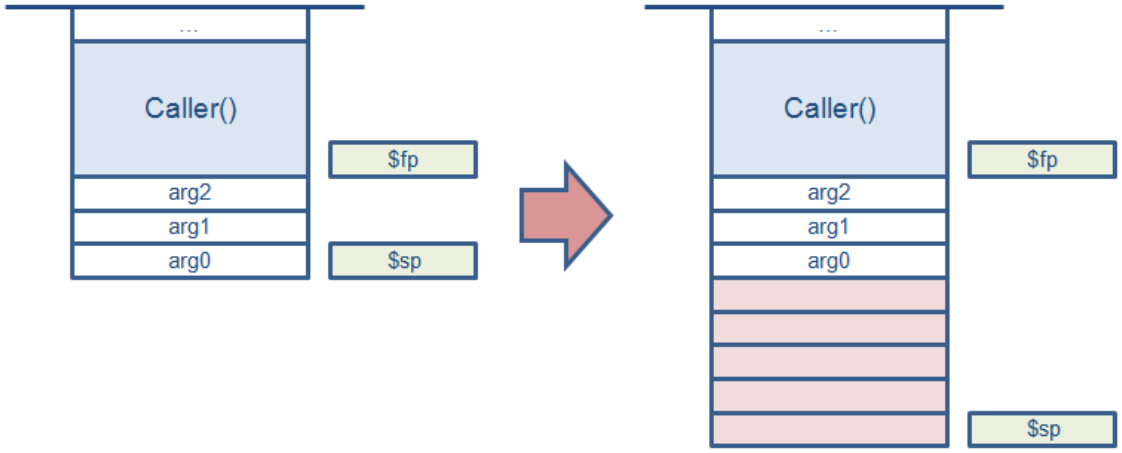
A continuación se ejecutan las instrucciones necesarias para calcular los valores de los argumentos de llamada y se almacenan éstos en posiciones crecientes a partir del *stack pointer*. Este último almacenamiento corresponde a la traducción de la instrucción de código intermedio PARAM. Desde el punto de vista de la función llamadora los argumentos se almacenan en orden inverso, es decir, el primer argumento se coloca en la posición más alejada del registro de activación de la función llamadora.



1.3 El proceso de llamada a una función

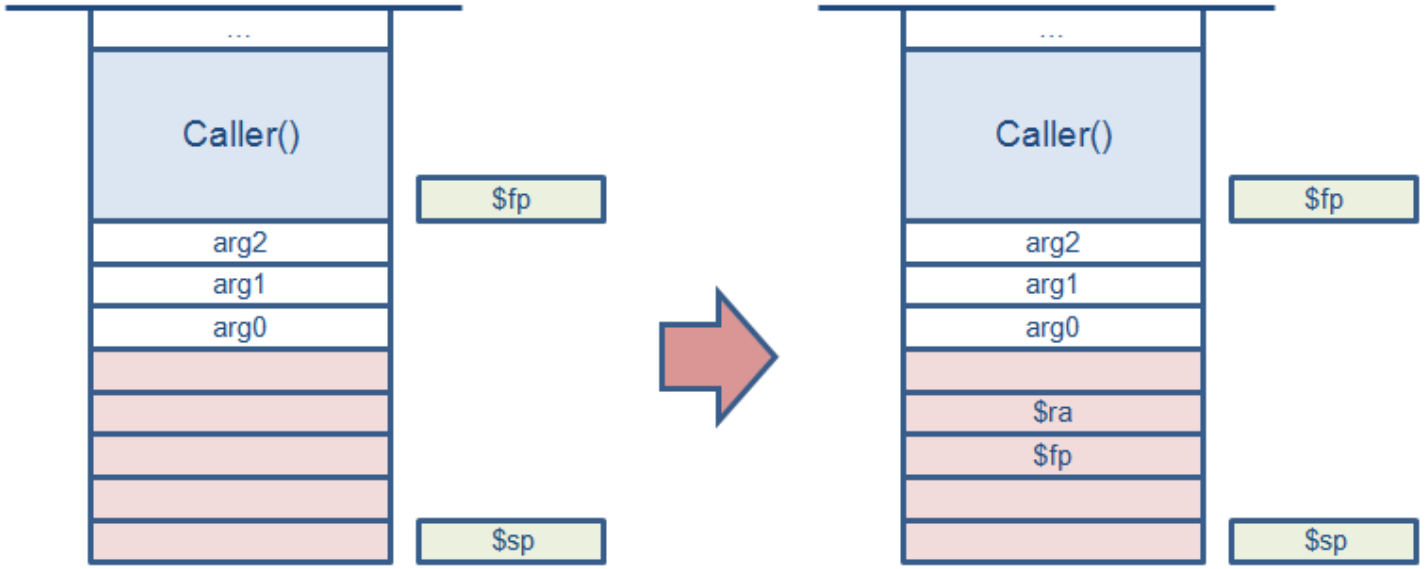
- Una vez almacenado el valor de los argumentos, se ejecuta la llamada a la función, que corresponde a la traducción de la instrucción de código intermedio CALL. Esto significa que se ejecuta un salto hacia la instrucción de comienzo de la función destinataria, que toma el control de la ejecución. Para realizar este salto se utiliza la instrucción de ensamblador JAL (jump and link). Esta instrucción realiza un salto y al mismo tiempo almacena en el registro $ra la dirección de la instrucción que se encuentra detrás del salto. El registro $ra (return address) corresponde al registro de propósito general $31.

1.3 El proceso de llamada a una función

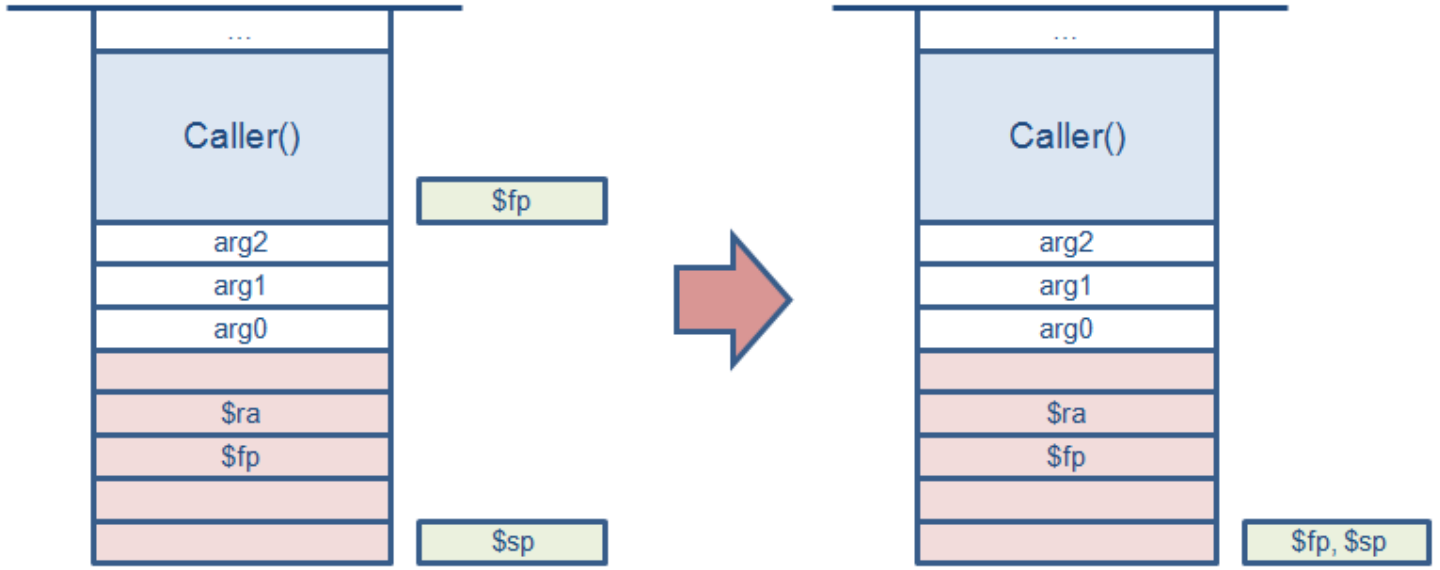


La primera acción de la función destinataria consiste en reservar espacio en la pila para su registro de activación. Para ello desplaza el *stack pointer* una cantidad correspondiente al tamaño del registro de activación.

1.3 El proceso de llamada a una función



A continuación almacena los valores de los registros *stack pointer* ($fp) y *return address* ($ra) en el registro de activación de la función receptora.



1.3 El proceso de llamada a una función

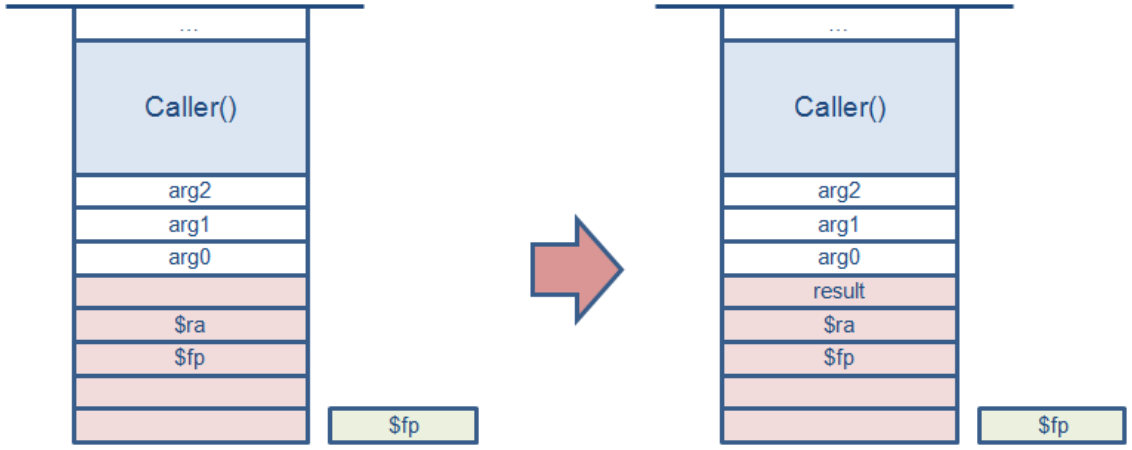
Por último, se asigna al *frame pointer* el valor de la dirección base del nuevo registro de activación, que corresponde al valor que tiene en este instante el *stack pointer*. A partir de este momento se ejecutan las instrucciones correspondientes al código de la función destinataria de la llamada.

1.4 El proceso de retorno de una función

- El proceso de retorno de una función se produce cuando se alcanza el final de la función o una instrucción return. Este proceso supone la destrucción del registro de activación de la función, el almacenamiento del resultado de la función en una posición determinada, la reactivación del registro de la función llamadora (caller) y la devolución del control de ejecución al código de la función llamadora.

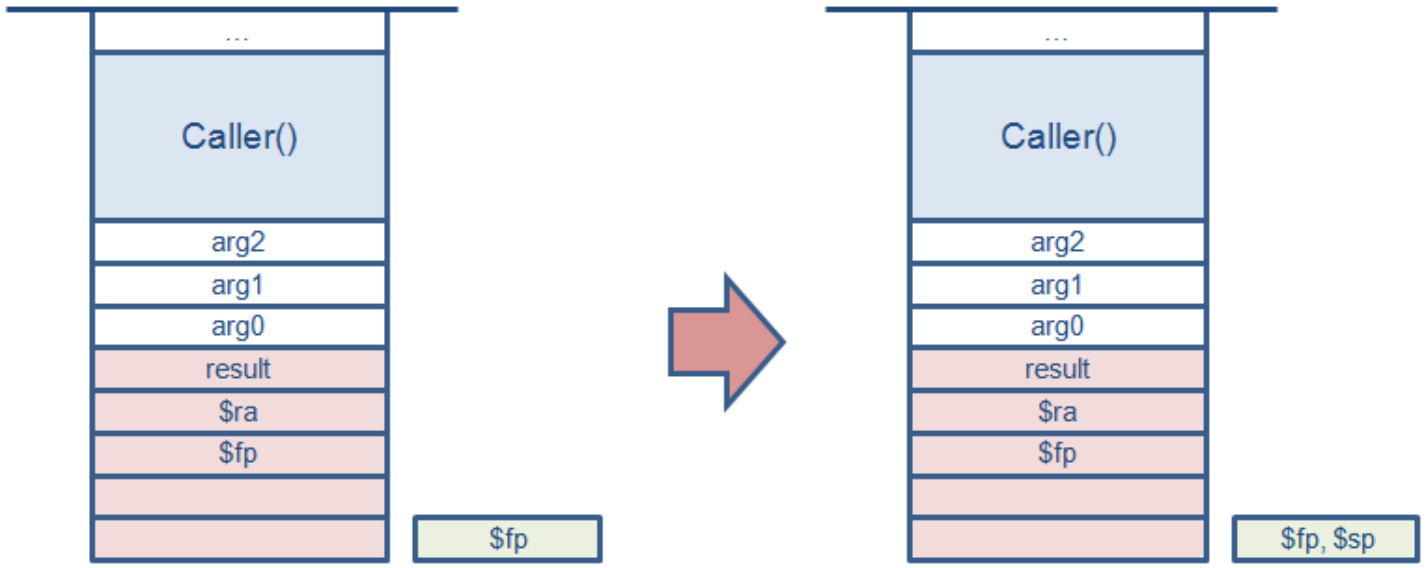
- El proceso de retorno que desarrolla el compilador de Tinto se compone de los siguientes pasos:

1.4 El proceso de retorno de una función



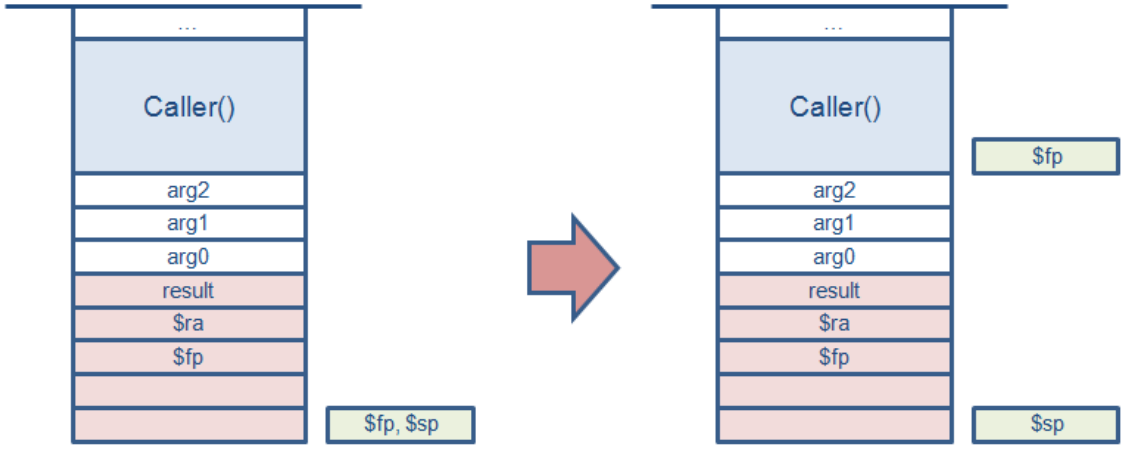
Almacenar el resultado de la función en la posición más alta del registro de activación (justo debajo de los argumentos).

1.4 El proceso de retorno de una función



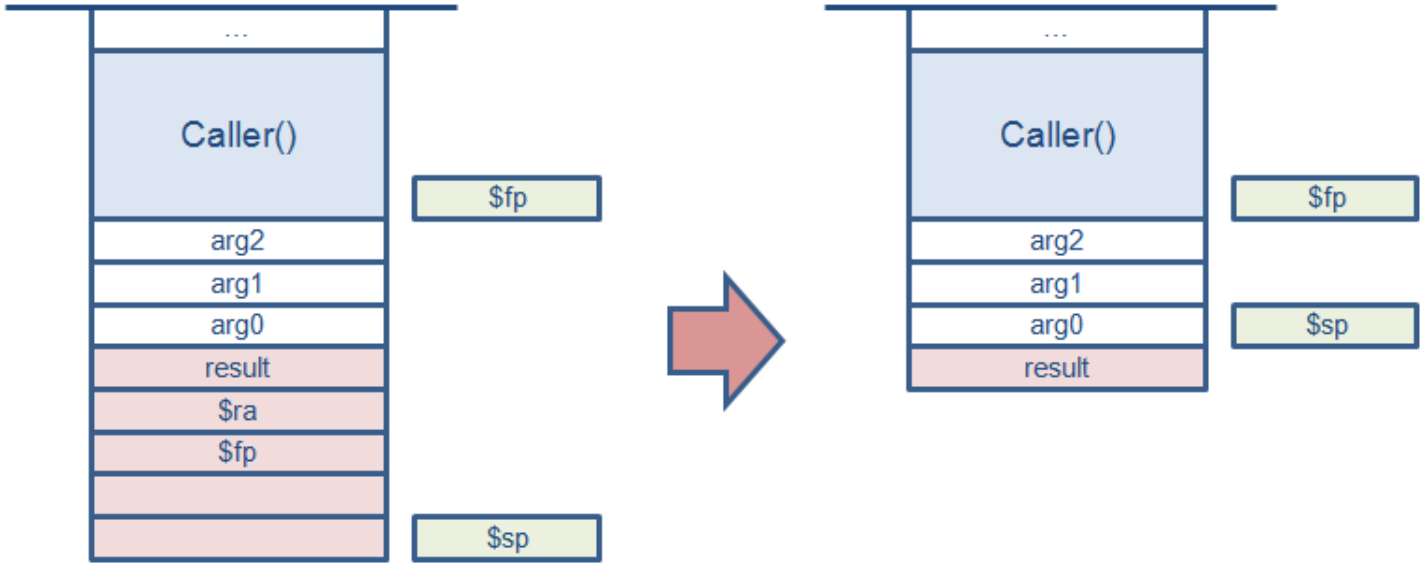
Colocar el *stack pointer* en la base del registro de activación (es decir, darle el mismo valor que el *frame pointer*).

1.4 El proceso de retorno de una función



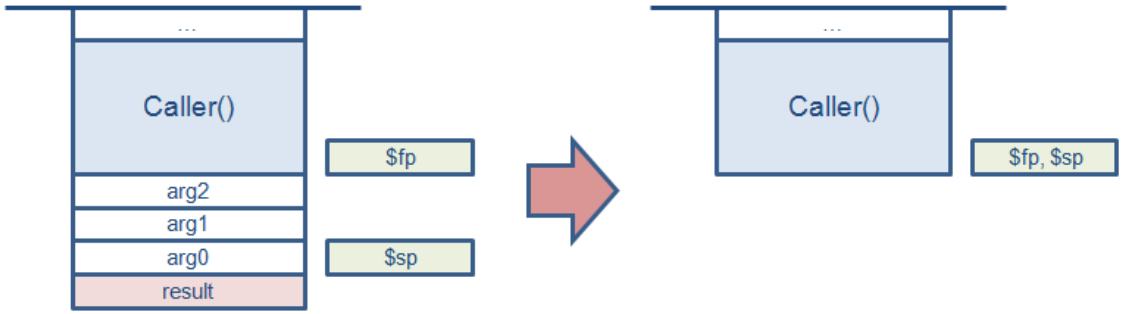
Restaurar los valores de los registros $fp y $ra almacenados en el registro de activación de la función. Es decir, colocar el *frame pointer* en la base del registro de activación de la función llamadora y almacenar en el registro $ra la posición de la instrucción de la función llamadora a ejecutar después de la llamada.

1.4 El proceso de retorno de una función



Desapilar el registro de activación de la función, desplazando el *stack pointer* la cantidad correspondiente al tamaño del registro.

1.4 El proceso de retorno de una función



- Saltar a la instrucción indicada en el registro $ra. En este momento el control de la ejecución vuelve a la función llamadora.
- Almacenar el resultado de la llamada en la variable correspondiente. Este valor se encuentra almacenado en la posición ($sp-4).
- Desplazar el stack pointer liberando el espacio reservado para los argumentos de la llamada. A partir de este momento la memoria vuelve a estar en el mismo estado que estaba antes de la llamada pero con el resultado de la llamada almacenado en una variable local o temporal.