A partir de esta sesión comenzaremos a estudiar una serie de convenios que suelen aplicarse en todos los sistemas basados en MIPS. Es necesario tener en cuenta que la mayoría de los convenios de los que hablaremos no suelen ser impuestos vía hardware, siendo responsabilidad de los programadores seguir dichas normas, de forma que el código escrito por diferentes personas pueda ser compatible y haga un uso efectivo del hardware MIPS.

GESTIÓN DE SUBRUTINAS:

Una subrutina no es más que un conjunto de instrucciones separadas del programa principal que realizan una determinada tarea y que puede ser invocada desde cualquier punto del programa.

Las subrutinas nos permiten estructurar un problema largo y complejo en subproblemas más sencillos, proporcionando una mayor facilidad a la hora de escribir, depurar y probar cada uno de los problemas por separado. De igual forma, si una tarea debe realizarse en varios puntos del programa, no es necesario replicar el código, sino que podemos hacer uso de las llamadas a estas subrutinas. Por último, subproblemas que suelan aparecer con frecuencia en el desarrollo de programas pueden ser implementados como subrutinas y agruparse en lo que llamamos bibliotecas (libraries) de forma que, cuando un programador quiera resolver un determinado problema y implementado, le basta con recurrir a una determinada biblioteca en invocar a la subrutina adecuada (reutilización de código).

Para que el uso de las subrutinas sea eficiente, un procesador debe proporcionar herramientas eficientes para facilitar las siguientes acciones: **llamada a una subrutina**; **paso de parámetros**; **devolución de resultados**; **continuación de la ejecución** del programa a partir de la siguiente instrucción a la que invocó a la subrutina.

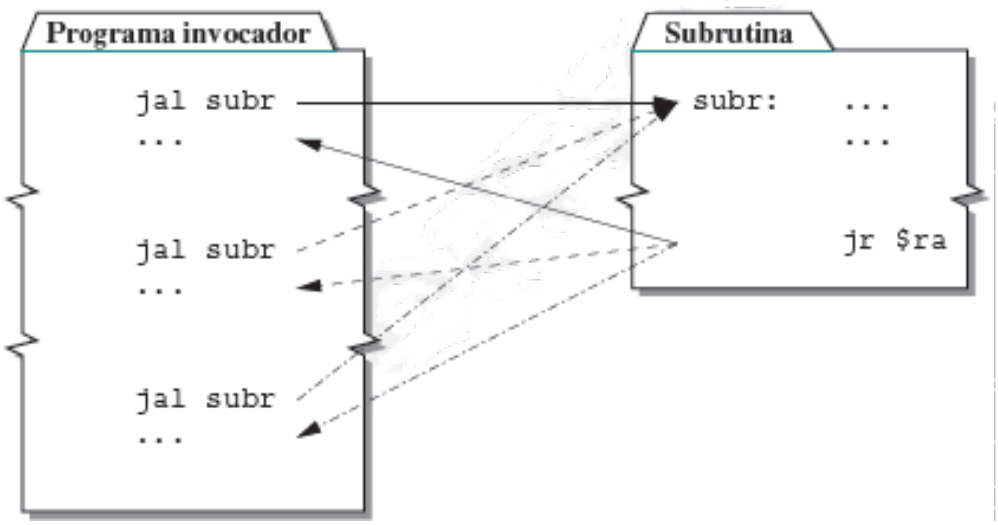
Llamada y retorno de una subrutina:

En el repertorio de instrucciones de MIPS32 disponemos de 2 instrucciones para gestionar la llamada y el retorno de una subrutina:

- **<< jal etiqueta >>** se utiliza el el programa invocador para llamar a la subrutina que comienza en la dirección de memoria indicada por la etiqueta. Como todos sabemos ya, la ejecución de la instrucción jal conlleva las siguientes acciones:
    - Almacenar la dirección de memoria de la siguiente instrucción a la que contiene a la instrucción jal en el registro $ra (return address): $ra ← PC+4.
    - Se lleva el control del flujo del programa a la dirección indicada en el campo etiqueta, o lo que es lo mismo, se realiza un salto incondicional a la dirección especificada por etiqueta: PC ← etiqueta.

- **<< jr $ra >>** se emplea en la subrutina para retornar al programa invocador. Esta instrucción realiza un salto incondicional a la dirección contenida en el registro $ra: PC ← $ra.

Así pues, si utilizamos correctamente estas dos instrucciones, podemos realizar de forma sencilla la llamada y retorno de desde una subrutina. El programa invocador debe llamar a la subrutina utilizando la instrucción jal. Esta instrucción almacena en $ra la dirección de vuelta y salta a la dirección indicada por etiqueta. Por último, cuando finalice la subrutina, ésta debe ejecutar la instrucción jr para retornar al programa que lo invocó. **Este mecanismo funcionará siempre que no se altere el contenido del registro $ra durante la ejecución de la subrutina.**



Paso de parámetros:

El mecanismo mediante el que el programa invocador y la subrutina intercambian información, recibe el nombre de **paso de parámetros.**

Los parámetros intercambiados entre el invocador y la subrutina pueden ser de 3 tipos según la dirección en la que se transmita la información: de **entrada**, de **salida** o de **entrada/salida.**

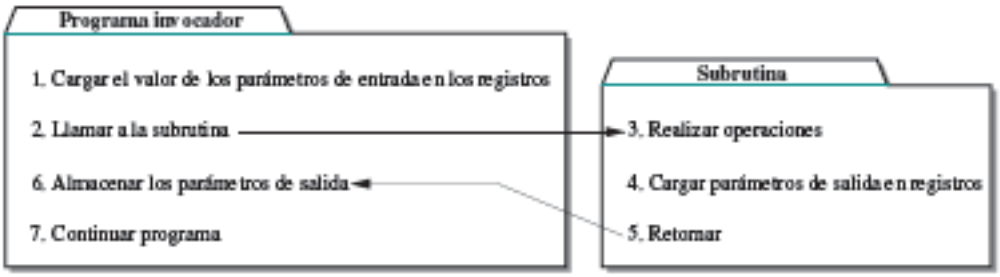
Entrada: proporcionan información del programa invocador a la subrutina.
Salida: devuelven información de la subrutina al programa invocador.
Entrada/Salida: proporcionan información del programa invocador a la subrutina y devuelven información de la subrutina al invocador.

Para realizar el paso de parámetros es necesario disponer de algún lugar físico donde almacenar y leer la información. Existen dos opciones: utilizar registros o utilizar la pila de memoria. Utilizar una u otra opción dependerá de la arquitectura en cuestión y del convenio para el paso de parámetros que se haya establecido. En esta sesión nos centraremos en el paso de parámetros por registro, dejando para la siguiente el uso de la pila de memoria.

MIPS32 establece un convenio de paso de parámetros mediante registros: para los registros de **entrada** se deben utilizar los registros **$a0, $a1, $a2 y $a3**; para los parámetros de **salida** se deben utilizar los registros **$v0** y **$v1.**

La transferencia de un parámetro puede realizarse de dos maneras diferentes: **por valor** o **por referencia.**

Por valor: se dice que un parámetro se pasa por valor cuando lo que se transfiere es el dato en sí.



Por referencia: se dice que un parámetro se pasa por referencia cuando se transfiere la dirección de memoria en la que se encuentra el dato.