INGENIERÍATÉCNICA EN INFORMÁTICA DE GESTIÓN. CURSO 2001-2002

LABORATORIO DE ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES

Sesión 13
Gestión de subrutinas

El diseño de un programa que resuelve un determinado problema puede simplificarse si se plantea adecuadamente la utilización de subrutinas. Éstas permiten dividir un problema largo y complejo en subproblemas más sencillos o módulos, más fáciles de escribir, depurar y probar que si se aborda directamente el programa completo. De esta forma se puede comprobar el funcionamiento individual de cada rutina y, a continuación, integrar todas en el programa que constituye el problema global de partida. Otra ventaja que aporta la utilización de subrutinas es que en ocasiones una tarea aparece varias veces en el mismo programa; si se utilizan subrutinas, en lugar de repetir el código que implementa esa tarea en los diferentes puntos, bastará con incluirlo en una subrutina que será invocada en el programa tantas veces como sea requerida. Yendo más lejos, subproblemas de uso frecuente pueden ser implementados como rutinas de utilidad (librerías), que podrán ser invocadas desde diversos módulos.

Con el fin de dotar de generalidad a una subrutina, ésta ha de ser capaz de resolver un problema ante diferentes datos que se le proporcionan como parámetros de entrada cuando se la llama. Por tanto, para una correcta y eficaz utilización de las subrutinas es necesario tener en cuenta, por un lado, la forma en que se realizan las *llamadas* y, por otro, el *paso de parámetros*.

Este boletín de prácticas comienza con la presentación de algunos ejercicios sencillos que permitan familiarizarse con el manejo de la pila, estructura imprescindible para una buena gestión de las subrutinas. Posteriormente se plantea la forma de invocar a una subrutina, el paso de parámetros a ésta y la gestión de variables que ésta emplea.

Gestión de la pila

Una pila es una estructura de datos caracterizada porque el último dato que se almacena es el primero que se obtiene después (esto es, una cola LIFO –"Last Input First Output"-). Para gestionar la pila se necesita un puntero a la última posición ocupada de la misma, con el fin de conocer dónde se tiene que dejar el siguiente dato a almacenar, o para saber dónde están situados los últimos datos almacenados en ella. Para evitar problemas, el puntero de pila siempre debe estar apuntando a una palabra de memoria. Por precedentes históricos, el segmento de pila siempre "crece" de direcciones superiores a direcciones inferiores. Las dos operaciones más típicas a realizar sobre esta estructura son:

- Transferir datos hacia la pila (apilar o, en inglés, "push"). Antes de añadir un dato a la pila se tienen que restar 4 unidades al puntero de pila.

- Transferir datos desde la pila (desapilar o, en inglés, "pop"). Para eliminar datos de la pila, después de extraído el dato se tienen que sumar 4 unidades al puntero de pila.

El ensamblador del MIPS tiene reservado un registro, `$sp`, como puntero de pila ("stack pointer"). Para realizar una buena gestión de la pila será necesario que este puntero sea actualizado correctamente cada vez que se realiza una operación sobre la misma.

El siguiente fragmento de código muestra cómo se puede realizar el apilado de los registros `$t0` y `$t1` en la pila.

```
          .text
main:     li    $t0,10
          li    $$t1, 13          # inicializar reg. $t0,$t1
          addi  $sp, $sp, -4      # actualizar $sp
          sw    $t0, 0($sp)       # apilar $t0
          addi  $sp, $sp, -4      # actualizar $sp
          sw    $t1, 0($sp)       # apilar $t1
```

Edita este programa, reinicializa el simulador y carga el programa.

Cuestión 1: Ejecuta el programa paso a paso y comprueba en qué posiciones de memoria, pertenecientes al segmento de pila se almacena el contenido de los registros `$t0` y `$t1`.

Cuestión 2: Modifica el programa anterior para que en lugar de actualizar el puntero de pila cada vez que se pretende apilar un registro en la misma, se realice una sola vez al principio y, después, se apilen los registros en el mismo orden.

Llamada y retorno de una subrutina

El juego de instrucciones del MIPS R2000 dispone de una instrucción específica para realizar la llamada a una subrutina, `jal etiqueta`. La ejecución de esta instrucción conlleva dos acciones:

- Almacenar la dirección de memoria de la siguiente palabra a la que contiene la instrucción `jal` en el registro `$ra`.
- Llevar el control de flujo de programa a la dirección `etiqueta`**.**

Por otra parte, el MIPS R2000 dispone de una instrucción que facilita el retorno de una subrutina. Esta instrucción es `jr $ra`, instrucción de salto incondicional que salta a la dirección almacenada en el registro `$ra`, que justamente es el registro donde la instrucción `jal` ha almacenado la dirección de retorno cuando se ha hecho el salto a la subrutina.

Para ejecutar cualquier programa de usuario el simulador XSPIM hace una llamada a la rutina `main` mediante la instrucción `jal main`**.** Esta instrucción forma parte del código que añade éste para lanzar a ejecución un programa de usuario. Si la etiqueta `main` no está declarada en el programa se genera un error. Esta etiqueta deberá siempre referenciar la primera instrucción ejecutable de un programa de usuario. Para que cualquier programa de usuario termine de ejecutarse correctamente, la ultima instrucción ejecutada en éste debe ser `jr $ra` (salto a la dirección almacenada en el registro `$ra`), que devuelve el control a la siguiente instrucción desde donde se lanzó la ejecución del programa de usuario. A partir de este punto se hace una llamada a una función del sistema que termina la ejecución correctamente.

El siguiente código es un programa que realiza la suma de dos datos contenidos en los registros `$a0` y `$a1` y la almacena en el registro `$v0`.

```
        .text
main:   li    $a0,10
        li    $a1,20
        add   $v0,$a0,$a1
        jr    $ra
```

Reinicializa el simulador, carga el programa y ejecútalo paso a paso, contestando a las siguientes cuestiones:

Cuestión 3: ¿Cuál es el contenido del PC y del registro `$ra` antes y después de ejecutar la instrucción `jal main`?

Cuestión 4: ¿Cuál es el contenido de los registros PC y `$ra` antes y después de ejecutar la instrucción `jr $ra`?

Cuestión 5: Reinicializa el simulador, carga el programa de nuevo y ejecútalo todo completo. Comprueba que la ejecución termina correctamente sin que salga el mensaje de error que salía en ejecuciones anteriores cuando no se incluía la instrucción `jr $ra`.

El siguiente programa realiza la misma operación de suma que en el ejemplo anterior pero implementando ésta en una subrutina:

```
        .text
main:   li    $a0,10
        li    $a1,20
        jal   subr
        jr    $ra
subr:   add   $v0,$a0,$a1
        jr    $ra
```

Reinicializa el simulador y carga este nuevo programa.

Cuestión 6: ¿Qué ocurre al ejecutar el programa?

Cuando una subrutina llama a otra utilizando la instrucción `jal` se modifica automáticamente el contenido del registro `$ra` con la dirección de retorno (dirección de memoria de la siguiente palabra a la que contiene la instrucción que ha efectuado el salto, es decir, la dirección de la instrucción posterior al salto). Esto hace que se pierda cualquier contenido anterior que pudiera tener este registro, que podría ser a su vez otra dirección de retorno, suponiendo que se llevan efectuadas varias llamadas anidadas. Así pues, en cada llamada a una subrutina se modifica el contenido del registro `$ra`, y sólo se mantiene en este registro la dirección de retorno asociada a la "última" llamada.

Una de las soluciones a este problema es que antes de ejecutar una llamada desde una subrutina a otra se salve el contenido del registro `$ra` en la pila. Como la pila crece dinámicamente, las direcciones de retorno de las distintas llamadas anidadas quedarán almacenadas a medida que éstas se van produciendo. La última dirección de retorno apilada estará en el tope de la pila, y ésta es justamente la primera que se necesita recuperar. Así pues, una pila es la estructura adecuada para almacenar las direcciones de retorno en llamadas anidadas a subrutinas.

El siguiente código modifica el programa del ejemplo anterior para que no se pierda la dirección de retorno del programa `main` con la llamada a la subrutina `subr` (esto es, con la instrucción `jal subr`) y sea posible la vuelta a la instrucción siguiente desde donde se hizo la llamada a la rutina

`main`. En la subrutina `subr`, como no se hace ninguna llamada a otra subruitna, no se necesita apilar el contenido del registro `$ra`.

```
        .text
main:   addi  $sp,$sp,-4
        sw    $ra,0($sp)
        li    $a0,10
        li    $a1,20
        jal   subr
        lw    $ra,0($sp)
        addi  $sp,$sp,-4
        jr    $ra
subr:   add   $v0,$a0,$a1
        jr    $ra
```

Reinicializa el simulador y carga el nuevo programa.

Cuestión 7: Comprueba qué contiene el registro `$ra` y el PC antes y después de ejecutar la instrucción `jal main`.

Cuestión 8: Comprueba el contenido de los registros `$ra` y PC antes y después de ejecutar la instrucción `jal subr`.

Cuestión 9: Comprueba el contenido de los registros `$ra` y PC antes y después de ejecutar la instrucción `jr $ra` que está en la subrutina `subr`.

Cuestión 10: Comprueba el contenido de los registros `$ra` y PC antes y después de ejecutar la instrucción `jr $ra` que está en la subrutina `main`.

Paso de parámetros

A la hora de implementar una subrutina hay que decidir:

- Parámetros a pasar a la rutina.
- Tipo de parámetros, que pueden pasarse:
  - Por valor (si se pasa el valor del parámetro).
  - Por referencia (si se pasa la dirección de memoria donde está almacenado el parámetro).
- Lugar donde se van a pasar los parámetros:
  - En registros.
  - En la pila.

El ensamblador del MIPS establece el siguiente convenio para realizar el paso de parámetros:

- Los cuatro primeros parámetros de entrada se pasan a través de los registros `$a0 - $a3`. A partir del quinto parámetro se pasan a través de la pila.
- Los dos primeros parámetros de salida se devuelven a través de los registros `$v0 - $v1`.

El siguiente programa implementa la llamada a una subrutina y el código de la misma, que devuelve una variable booleana que vale 1 si una determinada variable está dentro de un rango y 0, en caso contrario. La descripción algorítmica del mismo se muestra a continuación:

```
PROGRAMA EJEMPLO-3
VARIABLES
ENTERO: rango1=10; rango2=50; dato=12; res;
    INICIO
      res=subr(rango1,rango2,dato);
    FIN

FUNCION subr(ENTERO: para1, para2, para3):ENTERO;
    INICIO
      Si ((para3>=para1)and(para3<=para2)) ENTONCES
            subr=1;
      Sino
            subr=0;
      FinSi
    FIN
```

La subrutina tendrá, pues, los siguientes parámetros:

- Parámetros de entrada:
  - Las dos variables que determinan el rango, pasadas por valor, a través de `$a0` y `$a1`.
  - La variable a estudiar (si está dentro del rango), pasada por valor, a través de `$a2`.
- Parámetros de salida:
  - Variable booleana que indica si la variable estudiada está o no dentro del rango, y devuelta por valor a través de `$v0`.

El listado del programa en ensamblador se muestra a continuación:

```
            .data
rango1:     .word  10
rango2:     .word  50
dato:       .word  12
res:        .space 1
            .text
main:       addi   $sp,$sp,-4
            sw     $ra,0($sp)          # apilar $ra
            lw     $a0,rango1($0)      # $a0=rango1
            lw     $a1,rango2($0)      # $a1=rango2
            lw     $a2,dato($0)        # $a2=dato
            jal    subr                # saltar a subr
            sb     $v0,res($0)         # res=v0
            lw     $ra,0($sp)
            addi   $sp,$sp,4           # desapilar $ra
            jr     $ra                 # finalizar programa
```

```
subr:      blt    $a2,$a0,sino       # si $a2<$a0 saltar a sino
           bgt    $a2,$a1,sino       # si $a2>$a1 saltar a sino
entonces:  addi   $v0, $0,1          # $v0=1
           j      finsi              # saltar a finsi
sino:      add    $v0,$0,$0          # $v0=0
finsi:     jr     $ra                # retornar
```

Reinicializa el simulador, carga el programa, ejecútalo y comprueba el resultado almacenado en la posición de memoria `res`. Contesta a las siguientes cuestiones:

Cuestión 11: Identifica las instrucciones que se necesitan en:

- El programa que hace la llamada para:
  a) La carga de parámetros en los registros.
  b) La llamada a la subrutina.
  c) El almacenamiento del resultado.
- La subrutina para:
  a) La lectura y procesamiento de parámetros.
  b) La carga del resultado en `$v0`.
  c) El retorno al programa que ha hecho la llamada.

A continuación se modifica el código anterior para que los parámetros de entrada a la subrutina `subr` se pasen por valor mediante la pila y el de salida se pase también a través de la pila pero por referencia. En este caso, la descripción algorítmica del programa es la que se muestra a continuación:

```
PROGRAMA EJEMPLO-4
VARIABLES
ENTERO: rango1=10; rango2=50; dato=12; res;
    INICIO
         subr(rango1,rango2,dato,res);
    FIN

    PROCEDIMIENTO subr(ENTERO: para1,para2;para3;
                       VAR ENTERO: para4);
    INICIO
         Si (para3>=para1)or(para3<=para2)) ENTONCES
              para4=1;
         Sino
              para4=0;
         FinSi
    FIN
```

Los pasos a realizar, tanto por el programa que hace la llamada como por la subrutina, son los siguientes:

- En el programa que hace la llamada:
  a) Cargar el valor de los parámetros de entrada en la pila.
  b) Llamada a la subrutina.

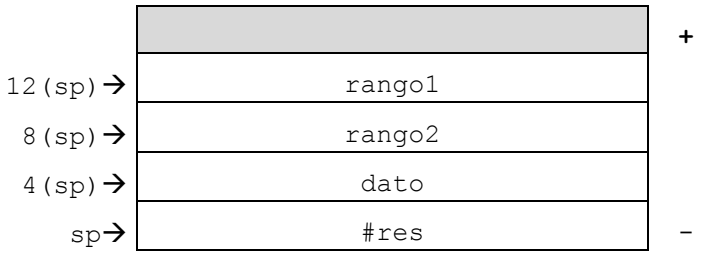
- En la subrutina:
  a) Lectura de los parámetros de entrada pasados a través de la pila.
  b) Procesamiento de los parámetros de entrada y generación del resultado.
  c) Almacenamiento del resultado en la dirección de memoria pasada a través del la pila.
  d) Retorno al programa que hizo la llamada.

El programa resultante en ensamblador del R2000 es el siguiente:

```
            .data
rango1:     .word  10
rango2:     .word  50
dato:       .word  12
res:        .space 1
            .text
main:       addi   $sp,$sp,-4
            sw     $ra,0($sp)          # apilar $ra
            lw     $s0,rango1($0)      # $s0=rango1
            lw     $s1,rango2($0)      # $s1=rango2
            lw     $s2,dato($0)        # $s2=dato
            la     $s3,res             # $s3=#res
            addi   $sp,$sp,-16         # reservar espacio parametros
            sw     $s0,12($sp)         # apilar rango1
            sw     $s1,8($sp)          # apilar rango2
            sw     $s2,4($sp)          # apilar dato
            sw     $s3,0($sp)          # apilar #res
            jal    subr                # saltar a subr
            addi   $sp,$sp,16          # limpiar parámetros
            lw     $ra,0($sp)
            addi   $sp,$sp,4           # desapilar $ra
            jr     $ra                 # finalizar programa

subr:       lw     $s0,12($sp)         # $s0=rango1
            lw     $s1,8($sp)          # $s1=rango2
            lw     $s2,4($sp)          # $s2=dato
            lw     $s3,0($sp)          # $s3=#res
            blt    $s2,$s0,sino        # si $s2<$s0 saltar a sino
            bgt    $s2,$s1,sino        # si $s2>$s1 saltar a sino
entonces:   addi   $s4,$0,1            # $s4=1
            j      finsi               # saltar a finsi
sino:       add    $s4,$0,$0           # $s4=0
            sw     $s4,0($s3)          # res=$s4
finsi:      jr     $ra                 # retornar
```

El siguiente esquema muestra la situación de la pila en el momento que empieza a ejecutarse la subrutina:



Por razones de simplicidad, en el ejercicio anterior se ha omitido apilar el valor de los registros que utiliza la subrutina. Esta operacón es obligatoria cuando queremos que no se pierda el contenido que tienen estos registros antes de ejecutar una subrutina ya que son modificados dentro de ésta. El apilado ha de realizarse al inicio de la subrutina y, justo antes del retorno al programa que la invocó, ha de realizarse la operación inversa de desapilado de dichos registros.

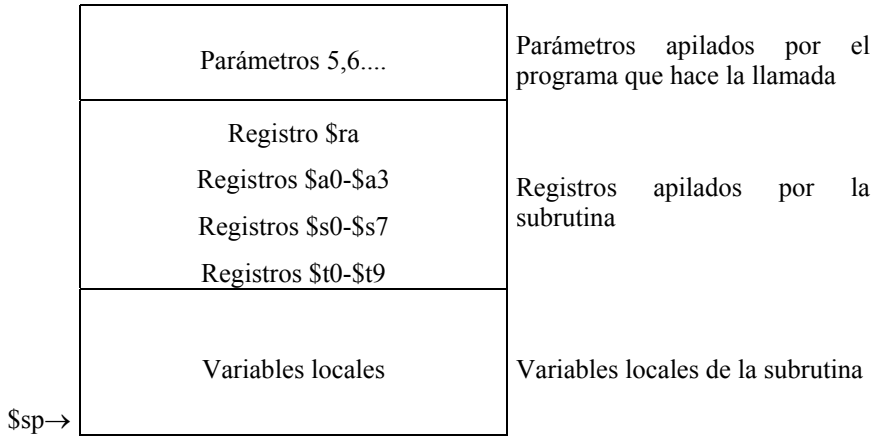
Ejecuta el programa anterior y comprueba que el resultado obtenido es el mismo que el del programa original.

Bloque de activación de la subrutina

El bloque de activación de la subrutina es el segmento de pila que contiene toda la información referente a la llamada a una subrutina (parámetros pasados a través de la pila, registros que modifica la subrutina y variables locales).

Los bloques de activación se pueden construir de diferentes formas. No obstante, lo que realmente importa es que el programa que hace la llamada y la subrutina deben estar de acuerdo en la secuencia de pasos a seguir.

La siguiente figura muestra un bloque de activación típico de una subrutina en la que el paso de parámetros se realiza siguiendo el convenio establecido en el MIPS R2000:



Problemas propuestos

1. Implementa una subrutina en ensamblador que calcule cuántos elementos de un vector de enteros de dimensión `n` son iguales a un elemento dado. Los parámetros de la subrutina serán los siguientes:

   - Parámetros de entrada:
     - Dirección del primer elemento del vector.
     - Número total de elementos del vector (dimensión).
     - Elemento a comparar.
   - Parámetro de salida:
     - Contador calculado.

   Realiza tres implementaciones diferentes en las que, respectivamente,:

   a) Todos los parámetros a la subrutina se pasen por valor (excepto aquéllos que obligatoriamente se deban pasar por referencia) y a través de registros.

   b) Todos los parámetros a la subrutina se pasen por referencia y a través de registros.

   c) Todos los parámetros que se pasen a la subrutina sean del mismo tipo que en el caso a) pero utilizando la pila como lugar de paso de parámetros.

2. Implementa una subrutina en ensamblador tal que, dado un vector de enteros de dimensión `n`, obtenga el elemento `i`-ésimo de dicho vector. La subrutina tendrá como parámetros de entrada: la dirección del vector, la dimensión del mismo y el índice del elemento a devolver. La subrutina devolverá como parámetro de salida el elemento i-ésimo del vector. Realiza la llamada y el retorno a la subrutina según el convenio establecido en el MIPS R2000.

3. Implementa una subrutina en ensamblador tal que, dada una matriz de enteros de dimensión `n x m`, almacenada por filas, obtenga el elemento `(i,j)` de dicha matriz. La subrutina tendrá como parámetros de entrada: la dirección de la matriz, las dimensiones de la misma y los índices del elemento a devolver. La subrutina devolverá como parámetro de salida el elemento `(i,j)` de la matriz. Realiza la llamada y el retorno a la subrutina según el convenio establecido en el MIPS R2000.