Organización y Arquitectura de Computadoras
2017-2
Práctica 6: Convención de llamadas a subrutinas

Profesor: José de Jesús Galaviz Casas
Ayudante de laboratorio: Roberto Monroy Argumedo

Publicada: Marzo 27, 2017.
Límite de entrega: Abril 1, 2017.

1. Objetivos

Generales:

- El alumno se familiarizará con los principios detrás de una invocación a una subrutina, analizando el flujo de las instrucciones y los recursos compartidos, haciendo énfasis en la interacción entre el hardware y el software.

Particulares:

Al finalizar la práctica el alumno:

- Conocerá una convencion de llamadas a subrutinas y de distribución de registros de la arquitectura MIPS.
- Tendrá la capacidad de escribir subrutinas usando dichas convenciones.

2. Requisitos

- **Conocimientos previos:**
  - Instrucciones aritméticas, lógicas y de manejo de flujo de programas del lenguaje ensamblador MIPS.
  - Modos de direccionamiento.
  - Programación recursiva.
- **Tiempo de realización sugerido:**
  5 horas.

- **Número de colaboradores:**
  Individual
- **Software a utilizar:**
  - *Java Runtime Environment* versión 5 o superior.
  - El paquete MARS [2].

3. Planteamiento

Una cualidad deseable del software es que sea reusable, por lo que comunmente un programa ejecuta en múltiples ocasiones una o más subrutinas. En un lenguaje de alto nivel, resulta transparente para el programador el proceso detrás de una llamada, ya que el compilador es el encargado de llevarlo a cabo, sin embargo, para que el programador de ensamblador pueda aprovechar eficientemente los registros y la memoria entre llamadas a subrutinas, además de asegurar la validez de los datos almacenados, es necesario emplear una convención que lo facilite. En esta práctica se escribirán programas que aprovechen la reutilización de código para practicar el uso de la convención de llamadas a subrutinas.

4. Desarrollo

Debido al limitado número de registros disponibles para el programador de ensamblador, cuando se realiza una llamada a una subrutina, es necesario liberar los registros para que puedan ser usados por ésta, por lo que los datos deben guardarse en memoria si serán usados después de la ejecución de la subrutina. El programador de ensambador es el encargado de realizar las acciones necesarias para preservar la información. No existe una única convención de llamadas a subrutinas para la arquitectura MIPS, la convención propuesta a continuación está basada en la convención descrita en [1].

4.1. Convención de registros

La convención asigna usos especificos para cada uno de los 32 registros con los que cuenta la arquitectura MIPS-32. La convención se define en la tabla 1 y los detalles serán cubiertos en el resto del desarrollo.

4.2. Pila de llamadas a subrutinas

Al bloque de memoria usado para preservar los datos entre llamadas, se le denomina **marco de llamada a subrutina** o simplemente marco. En éste, se pasan los valores de los argumentos para la subrutina en el caso de que los registros designados no hayan sido suficientes, se guardan los datos almacenados

Número	Nombre	Uso
0	`$zero`	Constante 0.
1	`$at`	Reservado para el ensamblador.
2 - 3	`$v0 - $v1`	Valores de retorno de una subrutina.
4 - 7	`$a0 - $a3`	Argumentos para una subrutina.
8 - 15	`$t0 - $t7`	Temporales. No preservados tras la llamada.
16 - 23	`$s0 - $s7`	Guardados. Preservados tras la llamada.
24 - 25	`$t8 - $t9`	Temporales. No preservados tras la llamada.
25 - 26	`$k0 - $k1`	Reservados para el sistema operativo.
28	`$gp`	Apuntador al área global.
29	`$sp`	Apuntador a la pila.
30	`$fp`	Apuntador al marco.
31	`$ra`	Dirección de retorno.

Tabla 1: Convención de uso de registros.

en registros que una subrutina pueda modificar pero que la rutina espera al regreso intactos y como espacio de memoria para variables locales.

Generalmente las llamadas a subrutinas se sirven de forma *LIFO*, *última entrada, primer salida*, de manera que resulta apropiado usar una pila para mantener los marcos, entonces es necesario reservar espacio en la memoria para ésta y dos registros para marcar los límtes del marco en el tope: `$sp` (*stack pointer*) el cual contiene la dirección de la primer palabra en el marco y `$fp` (*frame pointer*) con la dirección de la última palabra en el marco. La pila crece de las direcciones más altas de memoria a direcciones más bajas, cada marco es creado por una subrutina inmediatamente después que es invocada.

Un marco consta de seis secciones:

1. **Argumentos**. Si la subrutina invocará a otras subrutinas, se debe reservar espacio para el máximo número de argumentos que cualquiera de éstas requiera, el mínimo espacio que se reserva es para los cuatro registros de argumentos, sin importar que las subrutinas no los ocupen todos.

2. **Registros guardados**. La rutina invocadora espera intactos los datos almacenados en los registros guardados `$s0-$s7`, por lo que la subrutina debe respaldar los datos de los registros que serán modificados.

3. **Dirección de retorno**. Al finalizar la ejecución de la subrutina, ésta debe regresar el control a la rutina invocadora brincando a la dirección almacenada en `$ra`, por lo que si la subrutina invoca a otras subrutinas, debe guardar dicho valor ya que el registro será modificado en cada invocación.

4. **Apuntador a marco anterior**. La subrutina no conoce el tamaño del marco de la rutina invocadora, por lo que al actualizar el `$fp`, se pierde el apuntador al marco anterior, por lo cual, éste debe ser almacenado en el marco de la subrutina.

5. **Relleno**. El marco debe estar alineado a palabras dobles, ya que alguna subrutina puede necesitar guardar un dato de tamaño doble. Si la suma de los tamaños de los datos almacenados en las secciones anteriores no es múltiplo de ocho, se agrega un relleno con tamaño de una palaba.

6. **Variables locales**. Un espacio de memoria para guardar variables locales y los registros temporales `$t0-$t9`. El tamaño de esta sección debe ser múltiplo de 8.

Las secciones se colocan en el orden descrito partiendo del tope de la pila, ver figura 1.

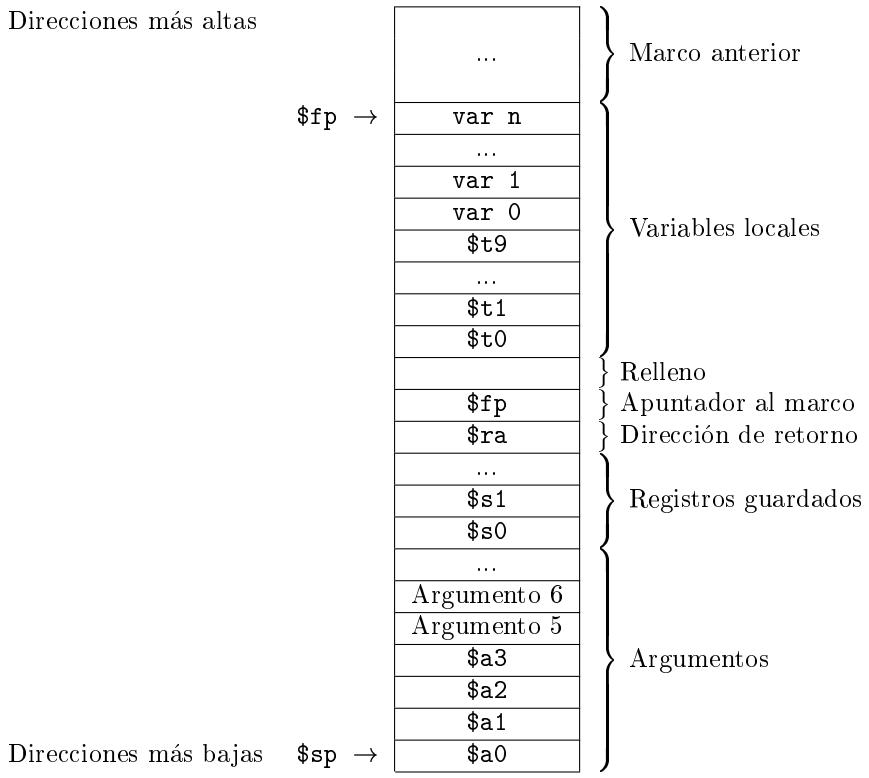

Figura 1: Secciones de un marco de llamada a subrutina.

Es importante notar que no siempre son necesarias todas las secciones en el marco.

4.3. Llamando a la subrutina

A manera de protocolo y concretando la convención, se presenta una serie de pasos que debe seguir tanto la rutina invocadora como la subrutina invocada, éstos se deviden en cuatro etapas.

4.3.1. Rutina invocadora

Invocación. La rutina invocadora debe:

1. **Pasar los argumentos a la subrutina**. Los primeros cuatro argumentos se pasan en los registros `$a0-$a3`, el resto en el espacio reservado para este fin en el marco de la rutina invocadora. No es necesario poner los primeros cuatro argumentos en la pila, pero sí reservar el espacio.
2. **Guardar registros temporales**. Si la rutina invocadora va a utlizar en el futuro los datos guardados en los registros temporales `$t0-$t9` o los argumentos `$a0-$a3`, los debe guardar en el marco, ya que pueden ser modificados por la subrutina.
3. **Pasar el control a la subrutina**. Ejecutar `jal` para brincar a la subrutina y guardar la dirección de retorno en el registro `$ra`, esta es la dirección de la siguiente instrucción que se ejecutará cuando la subrutina termine su ejecución.

Retorno. Al regresar de la subrutina, la rutina invocadorea debe:

1. Restaurar los registros temportales `$t0-$t9`.

4.3.2. Subrutina invocada

Preámbulo. Antes de realizar la tarea por la que fue llamada, la subrutina invocada debe:

1. **Reservar espacio en memoria para el marco**. Se actualiza el valor del `$sp`, restándole el tamaño del marco.
2. **Almacenar registros guardados**. Si la subrutina modifica los registros guardados `$s0-$s7`, debe almacenarlos en el marco.
3. **Guardar dirección de retorno**. Si la subrutina realiza llamadas a otras subrutinas, debe guardar el valor de `$ra`, el cual contiene la dirección de retorno de la rutina invocadora.
4. **Establacer el apuntador al marco**. Si es necesario modificar el `$fp`, se guarda el valor anterior en la pila y se actualiza con la suma del valor de `$sp` y el tamaño de la pila menos cuatro.

Conclusión. Al finalizar la tarea por la que fue llamada, la subrutina invocada debe:

1. **Retornar resultados**. Si se retornan valores, ponerlos en `$v0` y `$v1`.
2. **Restaurar registros guardados**. Restaurar los valores de los registros `$s0-$s7`, `$ra` y `$fp`.
3. **Eliminar el marco**. Se actualiza el valor de `$sp`, sumándole el tamaño de la pila.
4. **Regresar el control a la rutina invocadora**. Se brinca a la instrucción guardada en `$ra`.

5. Entrada

En cada ejercicio se debe implementar un programa distinto por lo que cada uno tiene una entrada distinta. Las entradas deben almacenarse en memoria y declararse al inicio del código fuente.

6. Salida

Al igual que en las entradas, cada programa tiene una salida distinta. Los resultados deben guardarse en el registro `$v0`.

7. Variables libres

No hay variables libres para el desarrollo de esta práctica.

8. Procedimento

Escribe las rutinas necesarias para cada uno de los ejercicios en lenguaje ensamblador de MIPS, usa la convención de llamadas a subrutinas y agrega una rutina *main* en donde se carguen de la memoria los argumentos en los registros adecuados y se llame a la subrutina que resuelva el ejercicio. Cada ejercicio debe encontrarse en un archivo de código fuente distinto. No olvides documentar el código.

9. Ejercicios

1. Completa el código de la figura 2 agregando los préambulos, invocaciones, conclusiones y retornos faltantes de las rutinas `main`, `mist_0` y `mist_1`. Los segmentos del código dado no pueden ser modificados. Responde en la documentación del código: ¿Qué hace la rutina?

2. Escribe una rutina que calcule recursivamente el coeficiente binomial de n en k utilizando la identidad de Pascal:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k},$$

para todo $k, n \in \mathbb{N}^+$, con casos base:

$\binom{n}{0} = 1$, con $n \geq 0$ y

$\binom{0}{k} = 0$, con $k > 0$.

```
        .data
a:      .word   5
b:      .word   4
        .text
main:   # Preambulo main
        # Invocacion de mist_1
        # Retorno de mist_1
        # Conclusion main
# mist_1 recibe como argumentos $a0 y $a1
mist_1: # Preambulo mist_1
        move    $s0, $a0
        move    $t0, $a1
        li      $t1, 1
loop_1: beqz$s0, end_1
        # Invocación de mist_0
        move    $a0, $t0        # Se pasa el argumento $a0
        move    $a1, $t1        # Se pasa el argumento $a1
        # Retorno de mist_0
        move    $t1, $v0
        subi    $s0, $s0, 1
        j       loop_1
end_1:  # Conclusion mist_1
        move    $v0, $t1        # Se retorna el resultado en $v0
# mist_0 recibe como argumentos $a0 y $a1
mist_0: # Preambulo mist_0
        mult    $a0, $a1
        # Conclusion mist_0
        mflo    $v0             # Se retorna el resultado en $v0
```

Figura 2: Programa misterio.

10. Preguntas

1. ¿Qué utilidad tiene el registro `$fp`? ¿Se puede prescindir de el?

2. Definimos como **subrutina nodo** a una subrutina que realiza una o más invocaciones a otras subrutinas y como **subrutina hoja** a una subrutina que no realiza llamadas a otras subrutinas.

   a) ¿Cuál es el tamaño mínimo que puede tener un marco para una subrutina nodo? ¿Bajo qué condiciones ocurre?

   b) ¿Cuál es el tamaño mínimo que puede tener un marco para una subrutina hoja? ¿Bajo qué condiciones ocurre?

3. Considera el siguiente pseudocódigo de la figura 3. En donde `a[5]` es un arreglo de tamaño 5 y "..." son otras acciones que realiza la rutina, además,

supón que en la `funcion_B` se realizan cambios en los registros `$s0, $s1` y `$s2`. Bosqueja la pila de marcos después del preámbulo de la `funcion_B`.

```
funcion_A(a, b)
    a[5]
    ...
    funcion_B(a, b, arreglo[0], arreglo[1], arreglo[2])
    ...
```

Figura 3: Rutina con llamado a subrutina.

11. Bibliografía

[1] David A. Patterson y John L. Hennessy. *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*. 5th. Morgan Kaufmann Publishers Inc., 2013.

[2] Kenneth Vollmar. *MARS MIPS Simulator*. Consultado: 8 de febrero, 2016. Publicado: 2014. URL: `http://courses.missouristate.edu/kenvollmar/mars/index.htm`.