UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Organización y Arquitectura de Computadoras
Practica 7

PRESENTA

Carlos Emilio Castañon Maldonado
Pablo César Navarro Santana

PROFESOR

José de Jesús Galaviz Casas

AYUDANTES

Ricardo Enrique Pérez Villanueva
Sara Doris Montes Incin
María Ximena Lezama Hérnandez

Organización y Arquitectura de Computadoras

Practica 7

Preguntas

1 **¿Cual es la diferencia entre las instrucciones *jal* y *jr*?**
La instrucción jal, que significa "jump and link", se utiliza para llamar a una subrutina o función en el programa, la instrucción jal toma una dirección de destino como argumento y la guarda en el registro de enlace antes de saltar a la dirección de destino, el registro de enlace se utiliza para almacenar la dirección de retorno después de que se haya completado la subrutina, esto permite que la subrutina retorne al punto de origen una vez que haya terminado su trabajo.

La instrucción jr, que significa "jump register", se utiliza para saltar a una dirección de destino almacenada en un registro específico, la instrucción jr toma un registro como argumento y salta a la dirección almacenada en ese registro, esto es útil cuando se quiere saltar a una dirección de destino que se ha calculado previamente y se ha almacenado en un registro.

La principal diferencia entre las instrucciones jal y jr es que jal se utiliza para llamar a una subrutina y guardar la dirección de retorno en el registro de enlace, mientras que jr se utiliza para saltar a una dirección de destino almacenada en un registro específico.

2 **¿Que utilidad tiene el registro $\$ra$? ¿Se puede prescindir de el?**
El registro $\$ra$ (link register) se utiliza para almacenar la dirección de retorno de una subrutina o función después de haber llamado a la misma con la instrucción jal, el registro $\$ra$ es importante para poder retornar correctamente a la instrucción que sigue inmediatamente después de la llamada a la subrutina.

Por ende el registro $\$ra$ es esencial para el correcto funcionamiento de las subrutinas, ya que sin el registro $\$ra$, no se podría retornar a la instrucción que sigue inmediatamente después de la llamada a la subrutina y por lo tanto, no se puede prescindir del registro $\$ra$ si se está trabajando con subrutinas o funciones.

3 **¿Que utilidad tiene el registro $\$fp$? ¿Se puede prescindir de el?**
El registro $\$fp$ (frame pointer), se utiliza para apuntar al inicio del marco de pila (stack frame) actual, recordemos que el marco de pila es una región de memoria reservada en la pila que se utiliza para almacenar variables locales y registros que deben ser restaurados después de la finalización de una función o subrutina, además debemos hacer énfasis en que el registro $\$fp$ se utiliza para referirse a las variables locales y los parámetros de la función, y también para realizar operaciones aritméticas en la pila, como mover el puntero de pila hacia arriba o hacia abajo, en otras palabras, el registro $\$fp$ ayuda a proporcionar un marco de referencia para el acceso a las variables locales y los parámetros de la función, lo que facilita la escritura de código.

Sin embargo con respecto a la pregunta sobre si se puede prescindir de el, depende del programa y la forma en que se manejan las variables locales y los parámetros de la función, en algunos casos, se puede prescindir del registro $\$fp$ y utilizar otros registros o técnicas de manejo de memoria para acceder a las variables locales y los parámetros de la función pero en otros casos este no es el caso.

4 **¿Como es que estas convenciones aseguran que la *subrutina invocada* (callee) no sobreescriba los registros de la *rutina invocadora* (caller)?**
Las convenciones de llamada a subrutinas (calling convention) aseguran que la subrutina invocada (callee) no sobrescriba los registros de la rutina invocadora (caller) al especificar qué registros deben ser preservados y cuáles pueden ser modificados.

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

➢ **¿Cual es el tamaño mınimo que puede tener un marco para una subrutina nodo? ¿Bajo que condiciones ocurre?**
En general, el tamaño mínimo de un marco para una subrutina nodo se determina por el número de registros salvados que se necesitan en el marco, por la convención de llamada a subrutinas, estas especifican que los registros $\$s0 - \$s7$ y el registro $\$fp$ deben ser preservados por la subrutina si los utiliza, si una subrutina nodo llama a otras subrutinas, es posible que necesite preservar otros registros adicionales, lo que aumentaría el tamaño del marco.

En cuanto a las condiciones en las que se produce el tamaño mínimo del marco, esto puede ocurrir cuando la subrutina utiliza pocos registros y no tiene argumentos o variables locales, en ese caso, el marco de la subrutina solo necesitará espacio para guardar los registros salvados y el registro $\$fp$, lo que resultará en un tamaño mínimo del marco, sin embargo, es importante tener en cuenta que este caso puede ser poco común, ya que la mayoría de las subrutinas nodos suelen tener argumentos y variables locales, lo que aumentaría el tamaño del marco.

➢ **¿Cual es el tamaño mínimo que puede tener un marco para una subrutina hoja? ¿Bajo que condiciones ocurre?**
El tamaño mínimo que puede tener un marco para una subrutina hoja es de 8 bytes, ya que se requiere al menos un espacio para almacenar la dirección de retorno y otro para guardar el registro $\$ra$.
Este tamaño se da en el caso en que la subrutina no tiene variables locales y no utiliza los registros $\$s0$ a $\$s7$, ya que estos también pueden ser utilizados para almacenar argumentos y valores temporales en la subrutina.

Si se utilizan registros adicionales para almacenar variables locales en la subrutina hoja, entonces el tamaño del marco será mayor, dependiendo de la cantidad de registros que se utilicen y su tamaño.

6 **Considera el siguiente pseudocódigo. 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] )
    ...
```

Para bosquejar la pila de marcos después del preámbulo de la función B, primero es necesario determinar qué registros se utilizan y deben ser preservados de acuerdo con la convención de llamada a subrutinas.
En este caso, la función B realiza cambios en los registros $\$s0$, $\$s1$ y $\$s2$, por lo que estos registros deben ser preservados en el marco de la función B.

Suponiendo que la función B utiliza la pila para almacenar las variables locales y que los argumentos de la función A y B también se almacenan en la pila antes de la llamada a la función B, la pila de marcos después del preámbulo de la función B se vería así:

Variable local	$\Leftarrow$ sp (puntero de pila)
Variable local	
Variable local	
$\$s2$	
$\$s1$	
$\$s0$	
Dirección de Retorno	
$arreglo[2]$	
$arreglo[1]$	
$arreglo[0]$	
Variable b	
Variable a	

Ejercicios

1 En el lenguaje de programación de tu elección, 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 caso base $\binom{n}{0} = 1$, con $n \geq 0$ y $\binom{0}{k} = 0$ con $k > 0$

En un archivo llamado `Practica7.C` :

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// Programa que calcula recursivamente el binomial de un numero
// Mediante la identidad de Pascal
int identidadPascal(int n, int k) {
    if (k == 0 || k == n) { // Caso base
        return 1;
    } else {    // Caso recursivo
        return identidadPascal(n-1, k-1) + identidadPascal(n-1, k);
    }
}

int main() {
    // Pedimos un valor de n al usuario
    int n;
    printf("Introduce un valor de n: ");
    scanf("%d", &n);

    // Pedimos un valor de k al usuario
    int k;
    printf("Introduce un valor de k: ");
    scanf("%d", &k);

    // Calculamos la identidad de Pascal
    int idPascal = identidadPascal(n, k);
    printf("La identidad de Pascal es ", n, k, idPascal);

    return 0;
}
```

Compilar con:

```
gcc -o Practica7 Practica7.c -lm
```

Resultados:

```
> ./Practica7
Introduce un valor de n: 10
Introduce un valor de k: 2
La identidad de Pascal es 45

> ./Practica7
Introduce un valor de n: 64
Introduce un valor de k: 3
La identidad de Pascal es 41664

> ./Practica7
Introduce un valor de n: 15
Introduce un valor de k: 9
La identidad de Pascal es 5005
```