Práctica 1: Sucesión de Fibonacci

Rolando Suarez
Jesus Muñoz

¿Cómo se implementa la recursividad en MIPS? ¿Qué papel juega la pila ($sp)?

La recursividad en MIPS se usa principalmente con el manejo del registro `$sp` (stack pointer). Este registro es un apuntador al tope de la pila. La pila se utiliza para guardar registros en memoria que corren el riesgo de ser sobrescritos cuando hacemos llamadas recursivas.

En MIPS hacemos llamadas recursivas como en cualquier otro lenguaje, es decir, llamando a la función que se está definiendo dentro de sí misma. Específicamente lo hacemos con la instrucción `jal` (jump and link), la cual genera un registro `$ra` que da el valor al PC indicado para retornar a la línea siguiente a la llamada de la función. Al final de la función debemos utilizar la instrucción `jr $ra` para retornar.

Precisamente para guardar este registro `$ra` en cada llamada recursiva, algo necesario para el retorno de cada llamada, se utiliza la pila y la modificación del registro `$sp` para abrir el espacio necesario. Se modifica el `$sp` tanto para guardar `$ra` como cualquier otro registro que utilice la función y que pueda ser modificado en cada llamada recursiva. Estos registros pueden ser, por ejemplo: `$a0`, `$a1`, o incluso temporales `$t0`.

—

¿Qué riesgos de desbordamiento existen? ¿Cómo mitigarlos?

Existen varios riesgos de desbordamiento al trabajar con recursividad y `$sp`. El primer riesgo puede deberse a hacer demasiadas llamadas recursivas que excedan la memoria, como por ejemplo el factorial de un número muy grande. El segundo riesgo se puede deber a no retornar el valor original del `$sp` al final de la función, lo que lo haría decrecer sin control en cada llamada. Este último se concatena con el error de no restaurar los registros almacenados en la pila, lo que puede generar corrupción de los mismos. Por último, un riesgo típico que existe es el de manejar mal el cálculo que se debe reservar en la pila. Hay que tomar en cuenta el tamaño en bytes de cada tipo de dato que vamos a guardar, por ejemplo 4 bytes para enteros.

—

¿Qué diferencias encontraste entre una implementación iterativa y una recursiva en cuanto al uso de memoria y registros?

Una vez sabes cómo usar la recursividad, esta implementación resulta más sencilla de entender y aplicar que la iterativa. Sin embargo, también implica utilizar el `$sp` y la pila en cada llamada para guardar los valores de los registros. El uso de la pila también implica limitaciones para hacer cálculos de Fibonacci para números muy grandes en el caso recursivo. Esto hace que el uso del método iterativo sea más seguro.

—

¿Qué diferencias encontraste entre los ejemplos académicos del libro y un ejercicio completo y operativo en MIPS32?

En los ejemplos de libro nos hemos enfocado más en el desarrollo de funciones o procedimientos puros. En los ejercicios completos tenemos, además de los procedimientos, una sección `.data` o `.text` donde debemos añadir mensajes para el usuario o los datos de entrada de nuestro programa. También debemos implementar un `main` para llamar a la función que estamos desarrollando y hacer las respectivas impresiones por pantalla.

—

Elaborar un tutorial de la ejecución paso a paso en MARS

Tutorial para Fibonacci iterativo en MIPS32:

Nuestro primer bloque de código, a partir de la línea 1, es para el mensaje que se va a dejar al usuario: “Ingrese un número: ”. Luego empezamos con el cuerpo principal de nuestro programa `main`. Dentro del cual, el primer bloque de código, en la línea 6, se usa para imprimir nuestro mensaje en pantalla. El bloque de la línea 10 se usa para guardar el número ingresado por el usuario en el registro de entrada `$a0` que utilizará luego nuestra función de Fibonacci. En la línea 14 llamamos nuestra función de Fibonacci con la instrucción de saltar y enlazar `jal`. Acá saltamos al cuerpo de nuestra función, que está a partir de la línea 23. Lo primero que hacemos en nuestra función es crear un registro para asignarle el valor de 1 (línea 24). Esto porque luego haremos dos verificaciones, para los casos en el que se trate de Fibonacci de 0 o de

1. En las siguientes operaciones utilizamos las funciones de `beq` para saltar al final de la función en los casos en que se trate del Fibonacci de 0, que es 0, y el de 1, que es 1. En caso de saltar a cualquiera de estos dos casos, lo que hace nuestra función es asignar el valor respectivo al registro de salida `$v0`, y retornar a la línea 15 con la instrucción `jr` y el registro `$ra`. En caso de que el dato de entrada `$a0` sea mayor que 1, seguiremos en la línea 29 dentro de nuestra función de Fibonacci. En este bloque lo que haremos será darle valor a tres registros necesarios para hacer el cálculo de Fibonacci, y a un último registro que hará el papel de nuestra variable iterativa `i` en nuestro ciclo `for`. El registro `$t1` será nuestro registro resultado, el que retornaremos culminado el ciclo. El registro `$t2` es el primer valor de la secuencia de Fibonacci y el registro que estará dos veces por detrás del valor resultado en la secuencia, es decir, Fib(n-2), y el registro `$t3` es el segundo valor de la secuencia y Fib(n-1). Sus valores iniciales son 0 y 1, respectivamente, y el valor inicial de `$t4` (nuestra "variable iterativa") será 1. Posteriormente, iniciamos el ciclo en la línea 34, considerando nuestra condición de parada cuando `$t4` sea igual o mayor que nuestro dato de entrada `$a0`. Lo que hará el ciclo es ir calculando el Fibonacci de todos los números anteriores en la secuencia hasta llegar al número de Fibonacci correspondiente en la posición de nuestro valor de entrada. Para esto, primero calcula el Fibonacci en el paso actual, que sería la suma de los dos números anteriores (`$t2` y `$t3`, primero y segundo de la secuencia) y los almacena en el registro `$t1`. Luego, avanzamos un paso en la secuencia, con `$t2` tomando el valor de `$t3` y `$t3` tomando el valor del actual `$t1` para el próximo cálculo. Finalmente, sumamos 1 a `$t4`, nuestra "variable iterativa", y saltamos de nuevo al inicio del ciclo para revisar la condición de parada con la instrucción `j`. En el siguiente bloque, en la línea 44, colocamos las instrucciones al final del ciclo, que es simplemente asignar nuestro Fibonacci calculado en `$t1` en el registro de salida `$v0`, y retornar a la línea 15 con la instrucción `jr` y el registro `$ra`. Finalmente, en la línea 16, usamos el bloque para imprimir en pantalla el resultado del cálculo. Y terminamos todo el programa con la instrucción `li $v0, 10`.

—

6. Justificar la elección del enfoque (iterativo o recursivo) según eficiencia y claridad en MIPS.

Nosotros escogimos el enfoque iterativo por ser más robusto frente a posibles errores de desbordamiento de pila cuando el número n del que se quiere calcular el Fibonacci es muy alto. Además, no vimos ninguna ventaja del uso de la recursividad sobre la iteración en este caso. Se puede lograr el mismo resultado incluso con mayor eficiencia utilizando la iteración, y se corren menos riesgos por no tener que utilizar ni preocuparse por la pila de memoria. Lo único que podemos ver positivo del uso de la recursividad es para usarlo como ejemplo ilustrativo de esta técnica de programación, ya que es un caso bastante

aclarador.