PRÁCTICA DE LABORATORIO 1: INFORME

Jose Contin V29.947.026
Luis Romero V26.729.561

Julio 2025

1 ¿Cómo se implementa recusividad en Mips32? ¿Qué papel cumple la pila ($sp)?

Se impleta mediante el uso la intruccion JAL que nos permite saltar a una etiqueta y guardar en $RA la direccion de regreso, luego utilizamos la instruccion JR a la cual le colocamos el registro $RA para regresar al punto donde se realizo la llamada.

Todo esto viendolo desde un punto de vista de un lenguaje de alto nivel seria como hacer que una funciona haga una llamada a si misma (cambiando los argumentos en cada llamada), lo cual es cosiderado recursividad. En detalle, la instruccion JAL salta a la etiqueta de la etiqueta del bloque de codigo donde fue llamada.

En recursividad, cada llamada debe preservar su estado para evitar sobrescribir datos (o entrar en un bucle infinito). Por lo tanto, También es necesario guardar el valor de estos registros en la pila $SP (stack) para almacenar el estado de los registro, preservar los valores de los argumentos y la direccion de retorno en cada llamada recursiva.

En general el proceso se divide de la siguiente manera:

1. Llamada recursiva con JAL
2. Retorno con JR
3. Uso de la pila ($sp)
4. Caso base y caso recursivo

2 ¿Qué riesgos de desbordamiento existen? ¿Cómo mitigarlos?

El riesgo existe cuando cuando se ingresa un valor mayor a 2^{32} (4.294.967.296, que se traduce en un rango de -2.147.483.648 a 2.147.483.647), esto porque todos los registros de Mips32 de propósito general son de 32 bits y por lo tanto, tienen un límite de almacenamiento fijo.

En nuestro programa esto ocurre cuando se introduce un valor mayor a 46, esto provoca que el valor de los registros sumados (en el caso iterativo, $t1 y $t0) supere el valor máximo que se puede almacenar en un registro de 32 bits (la suma de ambos al llegar a 47 es 2.971.215.073, lo cual es mayor al rango antes dicho), dando como resultado un desbordamiento aritmético.

Esto se puede mitigar de varias formas, una de ellas sería limitando el número máximo que se puede ingresar el programa, que en este caso sería 46, o simulando un registro de 64 bits dividiendo en dos registros de 32bits, aumentando asi la capacidad que este puede llegar a almacenar. También se pueden usar el coprocesador 1 (FPU) para almacenar enteros de 64 bits, pero no es adecuada para aritmética de enteros.

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

La diferencia principal radica en que al hacer la implementacion iterativa no es necesario aumentar el tamaño de la pila para luego almacenar cietos registros que luego necesitarían ser recuperados, algo que es vital para la implementación recursiva. Esto resulta en que, la implementación iterativa es más eficiente en memoria.

Además los registros que se utlizan en la versión iterativa son Registros fijos: $t0 (contador), $t1 y $t2 (valores anteriores de Fibonacci). Mientras que en el recursivo, son registros usados para argumentos ($a0, $a1, $a2) y temporales ($t0).

Por lo consideramos que es mucho mas facil implementarlo y depurarlo de forma iterativa. En terminos de redimiento la implemetancion recursiva deja mucho que desear, ya se necesita mucha mas memoria y ejecucion de operaciones.

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

En los ejemplos del libro, no se hace enfasis en la estructura general y fundamental de un programa, solo se muestran las secciones mas importantes que resalta el como realizar ciertas operaciones especificas, mas al realizar un ejercicio completo y operativo hay que tomar muchas variables en cuenta, desde el uso de Mars hasta la entrada y salida del programa pasando por limitacioes que pueda llegar a contener el emulador usado para ejecutar el codigo esamblador.

5 Elaborar un tutorial de la ejecución paso a paso en MARS.

- Paso 1: Descargar MARS 4.5 en página de la Universidad de Missouri y ejecutarlo en tu PC (se necesita java).
- Paso 2: Abrimos MARS 4.5 y cargamos un programa en MARS. Si queremos crearlo de cero, hay que hacer click en **File > New**, y escribirmos nuestro programa en la pestaña **Edit**. Si ya tenemos el archivo **.asm**, lo importamos dandole a **File > Open** y buscamos el archivo.
- Paso 3: Una vez tengamos listo el programa, lo guardamos, ya sea con **Crtl + S** o en el ícono de save, y lo ensamblamos dandole click al ícono de Assemble o con la tecla **F3**
- Paso 4: Luego, en la parte de abajo (donde se encuentra el cuadro de los mensajes de MARS), nos dirá el emsamblador si ha habido algún problema o no. Si todo ha ido bien debe haber salido un mensaje tipo "Assemble: operation completed successfully." y estaremos ahora en la pestaña de **Execute**, donde encontraremos las instrucciónes que va a seguir nuestro programa.

  Para ejecutar paso a paso nuestro programa, iremos a los íconos que aparecen al lado de el ícono de Assemble, y vemos que al principio hay dos opciones, o corremos el programa completo (dando click en el ícono o también con **F5**), o corremos paso por paso el programa (dando click en el ícono o también con **F7**). En nuestro caso, ejecutamos el correrlo de paso por paso.
- Paso 5: Veremos como se empezará a resaltar de amarillo las líneas de código que se estan ejecutando al momento, al igual que, se marcarán de verde los cambios que se hagan en los registros (en la pestaña de la derecha). También podemos devolvernos un paso hacia atras o backstep

(con el boton que está a la derecha de el de paso por paso o podemos usar **F8**). Al fnalizar, en los mensajes de MARS aparecerá "program is finished running". Para reiniciar la memoria y los registros, se pulsa el botón Reset (el botón más a la derecha del mismo panel de los otros botones de control o pulsamos **F12**.

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

Como dijimos antes, el enfoque iterativo nos parece el más acertado en este caso, por su eficiencia y simplicidad. En cuanto a eficiencia, siempre el iterativo será mejor, por un lado, en memoria ya que, usa registros fijos ($t0, $t1, etc.) y no depende de la pila (O(1) en espacio), algo que no se cumple en el enfoque recursivo (O(n) en espacio y riesgo de stack overflow). Por otro lado, en tiempo, que aunque ambos compartan el mismo O(n), el algorimo recursivo requiere alto costo por el manejo manual de la pila.

En cuanto a la legibilidad, el iterativo tiene un flujo lineal fácil de seguir (ideal para depuración) y no requiere manejo explícito de la pila. Aunque el recursivo es más fiel a la definición matemática (fib(n) = fib(n-1) + fib(n-2)), la recursión es menos intuitiva por el manejo manual de la pila y eso lo hace difícil de depurar (rastrear llamadas anidadas y estados de registros).

7 Análisis y Discusión de los Resultados

Ambos programas (iterativo y recursivo) calculan el n-ésimo número de Fibonacci, pero son diferentes en su implementación, eficiencia y corrección. Ambos comienzan leyendo un entero **n** y tienen una validación base (chequean si el número es negativo). Luego se chequea (fuera de la función pincipal de Fibonacci) si el número ingresado a sido un 1 o un 2, ya que en ambos casos, su fibonacci es igual a 1. Esto contaría como el "caso base" de ambos programas. Hasta aquí ambos programas son exactamente iguales.

Téngase en cuenta que, el valor que ingresa el usuario es sumado en 1. Esto se hace de tal forma ya que, queríamos que en nuestro programa existiera el fibonacci "0", y que este direse como resultado 0, de tal forma que el inicio del programa sea la suma entre fib(0) y fib(1). Si esto no fuese así, el resultado de fib(1) sería igual a 0.

Luego, llegamos a las funciones principales de cada uno, donde difieren de acuerdo a su implementación. En el caso iteratio, la función toma cuatro registros ($t0, $t1, $t2, $t3) y un registro de argumento ($a0). En los registros

$t1 y $t2 se guardan los sumandos de la sucesión de fibonacci, $t3 sirve como un registro temporal (guarda la suma de $t1 y $t2) y también alli se guarda el resultado y $t0 y $a0 sirven para controlar la cantidad de veces que se va a repetir el bucle. Este comienza en 3 (ya que los casos 0, 1 y 2 ya fueron chequeados antes) y se va sumando de 1 en 1 hasta el valor desado en $a0. Cuando $t0 es mayor que $a0 sale del bucle y se lo pasa como argumento a la función **mostrar** que muestra por pantalla el resultado.

Ahora, en el caso recursivo, la función toma también 3 registros de argumentos ($a0, $a1, $a2) y un registro temporal ($t0). Primero se reserva espacio en la pila y se guardan estos registros para que en cada llamada se conserven (incluyendo $ra que nos permite poder devolvernos). luego entra en la condicion (la cual funciona como el bucle en la forma iterativa) que es controlada por el valor en $a0 que va a ir disminuyendo de 1 en 1 hasta llegar a 3. si no se cumple la condicion, la función se llama a si misma cumpliendo con el método recursivo. Si se cumple la condición, entra en caso base, donde guarda el resultado en $v0 y se regresa, para recuperar la pila en cada llamada. Por último, el resultado se muestra por pantalla.