Informe de Práctica de Laboratorio I

José M. Díaz M. (25.682.785)
Sergio José Noguera (30.572.211)

08 de julio de 2025

1. ¿Cómo se implementa la recursividad en MIPS32? ¿Qué papel cumple la pila (*sp*)?

La recursividad, a nivel técnico, consiste en que una función se llame a si misma. Para resolver un problema, subdividiéndolo en problemas mas pequeños, las funciones cuentan con su caso base y su caso recursivo, esto mismo es aplicable en las llamadas a funciones de MIPS 32 gracias al uso de las pilas o stacks. Las características que debe tener una llamada recursiva para implementarla son las siguientes:

- Caso Base y Caso Recursivo: Toda función recursiva debe tener un caso base (condición de parada) que realiza los cálculos necesarios y coloca el resultado en el registro de valor de retorno ($v0) para saltar directamente al fin de la función para limpiar la pila y retornar. La función también debe de contar con un caso recursivo que llama a la función nuevamente con parámetros diferentes que permitan acercarse al caso base; igualmente, utiliza los valores de retorno ($v0) junto con el registro $ra (la dirección de retorno) que le permite volver a la función actual.

- La pila: esta es la estructura que cumple la función de guardar a través de la instrucción sw, los posibles registros $a0-$a3 (para argumentos), $v0-$v1 (para valores de retorno), $s0-$s7 (registros salvados), punteros de pila ($sp) y las más importante, $ra (la dirección de retorno) que permite que la función regrese correctamente a su llamador original.

- Epilogo o final de la función: Carga los valores originales de los registros que se guardaron en la pila utilizando la instrucción lw y finaliza la función con $ra para ir a la función o algoritmo que lo llamo en primera instancia.

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

Riesgos de desbordamiento

- Recursividad infinita: Es cuando una función recursiva nunca termina debido a que nunca alcanza el caso base para terminar o la profundidad de la recursión excede el de la pila, esto provoca que se agote la memoria de la pila asignada generando un error de segmento y posiblemente se cuelgue el programa.

- Grandes marcos de pila: Es cuando se agregan demasiadas variables, ya sea en la pila o en la función recursiva; en general, esto provoca que, en unas cuantas bajas profundidades de recursión, se desborde la pila provocando que se agote la memoria de la pila asignada, lo cual conllevaría a que se genere un error de segmento y posiblemente se cuelgue el programa.

- Desbordamiento de buffer en la pila: Es una vulnerabilidad de seguridad que ocurre cuando se escriben más datos en una zona de memoria reservada para la pila de lo que esta puede contener. Este tipo de vulnerabilidad es especialmente peligrosa porque puede provocar que los datos excedentes ternimen sobreescribiendo otras partes del marco de pila, incluyendo la dirección de retorno.

Como mitigar los riesgos de desbordamiento

- Tener bien asegurado el caso base: Para toda función recursiva es muy importante que pueda alcanzar siempre, a través del caso base, su condición de parada. Para esto, los parámetros de las llamadas recursivas deben modificarse de tal manera que siempre progresen hacia el caso base.

- Limitar la Profundidad de Recursión: Al establecer un límite, se deben evitar posibles problemas de memoria en la profundidad de la recursividad, y si la recursión excede ese límite, la función debería lanzar un error, mensaje o retornar un valor especial en lugar de seguir llamándose.

- Convertir la Recursividad en Iteración: Es bien sabido que hay funciones recursivas que se pueden reescribir de forma iterativa, de manera que sean más rápidas y con menor probabilidad de generar desbordamientos.

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

FUNCIONES ITERATIVAS

Diferencias en memoria:

- Las pilas en las funciones iterativas usan una cantidad de memoria constante o pequeña, independientemente del número de iteraciones.

- Solo tienen un marco de pila para cada función y las variables temporales o argumentos actualizan sus cambios con cada iteración. Estas funciones se actualizan en los mismos registros o en las mismas posiciones de memoria dentro del único marco de pila.

Diferencias en registros:

- Los registros como t0-$t9, $s0-$s7 se utilizan para almacenar contadores de bucle, valores intermedios y variables locales que siempre se reutilizarán en cada iteración. Estos valores no se sobrescriben, por lo que no es necesario guardar o restaurar sus valores.

- Los registros $a0-$a3 y Retorno $v0-$v1 se usan en la primera y en ultima iteración; al igual que los registros temporales, no hay llamadas repetidas que sobrescriban y necesiten guardar estos registros en la pila.

- El registro $ra se guarda una sola vez en la pila y se restaura una vez al final.

- El Puntero de Pila $sp solo se ajusta una vez al inicio para el marco de pila de la función y una vez al final.

FUNCIONES RECURSIVAS

Diferencias en memoria:

- Las pilas en las funciones iterativas usan una gran cantidad de memoria, debido a que cada llamada recursiva crea un nuevo marco de pila. La cantidad de memoria de pila utilizada es directamente proporcional a la profundidad de la recursión.

Diferencias en registros:

- Los registros $t0-$t9 y $s0-$s7 mantienen valores importantes que deben preservarse antes, durante y despues de las llamadas recursivas. Cuando una función es recursiva, estos registros pueden ser modificados y/o sobreescritos durante su ejecución, por lo que es necesario garantizar que se puedan guardar o restaurar sus valores originales, tanto al inicio como al final de cada llamada recursiva. Las instrucciones sw en el prólogo y lw en el epilogo, son vitales para mantener los valores, respectivamente.

- Los registros $a0-$a3 y Retorno $v0-$v1 a lo largo de las llamadas repetidas se sobrescribirán y necesitarán guardarse en la pila antes de la llamada y restaurarse después.

- El registro $ra debe ser guardado en la pila antes de cada llamada recursiva.

- El Puntero de Pila $sp se ajusta en cada llamada para asignar un espacio para el nuevo marco de pila y se ajustará de nuevo al final para liberar ese espacio.

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

ASPECTO DIFERENCIAL	EJEMPLOS DEL LIBRO	EJERCICIO COMPLETO
Objetivo Principal	Los ejemplos del libro se encargan de enseñar conceptos específicos de MIPS 32 para enteder los pasos e instrucciones de esta arquitectura	Un ejercicio completo debe ser capaz de desarrollar y mostrar un programa mucho mas íntegro y funcional, y debe cumplir con el trabajo complejo para el que fué creado.
Complejidad	Los ejemplos del libro muestran fragmentos simples y focalizados	Un ejercicio completo puede ser sinónimo de programas integrales con múltiples componentes
Contexto	Los ejemplos del libro se muestran aislados y controlados	Un ejercicio completo puede estar integrado con otros sistemas y rutinas
Manejo de Errores	Los ejemplos del libro tienen un manejo de errrores limitado o simplificado	Un ejercicio completo puede tener un un manejo de errrores implementado completamente
Optimización	Para los ejemplos del libro, esto es algo secundario, ya que lo que se busca no es un programa con buen rendimiento, sino uno que ayude al entendimiento	Para un ejercicio completo, esto es importante, ya que un buen rendimiento ayuda a que el programa realice mejor su trabajo
Documentación	Los ejemplos del libro tienen una documentación mínima, enfocada en conceptos	Un ejemplo completo puede tener una documentación más completa y profesional
Pruebas	Los ejemplos del libro se pueden usar para casos básicos	Un ejemplo completo puede usarse para una suite completa de pruebas
Interacción	Los ejemplos del libro pueden interactuar con componentes específicos	Un ejemplo completo puede interactuar con múltiples partes del sistema

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

Configuración Inicial

1 - Abrir MARS en la computadora

2 - Configurar la ventana principal

- Hay que asegurarse de tener visible el editor de texto.
- Verificar que las ventanas de registro estén accesibles.
- Confirmar que el panel de registros esté visible.

Pasos detallados de ejecución

1 - Creación del programa

- Hacer clic en "File", seguido de "New" o presionar Ctrl+N.
- Escribir el código MIPS en el editor.
- Guardar el archivo con extensión .asm.

2 - Compilación

- Presionar F3 o hacer clic en el botón "Assemble".
- Revisar la ventana de mensajes para detectar errores.
- Si hay errores, hay que correfirlos y volver a compilar.

3 - Preparación para la ejecución

- Colocar el cursor en la primera línea de código que se desea ejecutar.
- Establecer breakpoints si se necesita (F5).
- Preparar cualquier dato de entrada necesario.

4 - Ejecución del Programa

- Usar F5 para ejecutar todo el programa.
- Otra opción es usar F7 para ejecutar paso a paso.
- Observar los cambios en: Registros, Memoria y Pila.

Depuración

1 - Durante la ejecución

- Hay que monitorear los valores de los registros.

- Hay que verificar los cambios en la memoria.
- Hay que usar los breakpoints estratégicamente.

2 - Al encontrar errores

- Identificar dónde se produjo el error.
- Reviar los valores de los registros.
- Corregir el código y volver a compilar.

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

Usar Enfoque Iterativo cuando:

- Se trabaja con grandes conjuntos de datos.
- La memoria es un recurso limitado.
- Se requiere máxima eficiencia en el tiempo de ejecución.
- El problema tiene una naturaleza secuencial clara.

Usar Enfoque Recursivo cuando:

- El problema tiene una estructura jerárquica natural.
- La claridad del código es prioritaria sobre el rendimiento.
- Se trabaja con estructuras de árbol o grafo.
- La profundidad máxima de recursión es conocida y manejable.

En MIPS 32, el enfoque iterativo suele ser la opción más segura y eficiente para la mayoría de los casos, especialmente cuando se trabaja con conjuntos de datos grandes o se requiere optimización de rendimiento. Sin embargo, para problemas que tienen una estructura naturalmente recursiva (como árboles binarios o expresiones matemáticas), el enfoque recursivo puede proporcionar un código más claro y mantenible, siempre y cuando se controle cuidadosamente la profundidad de recursión.

7. Análisis y Discusión de los Resultados

Se puede decir que la recursividad y la iteratividad son dos formas muy distintas de llegar al mismo resultado en un algoritmo, ambas tienen ventajas y desventajas, siendo la función iterativa la que tiene más ventajas por encima de la recursiva, por cuestiones como su menor uso en la memoria de la pila, que es clave para poder llegar a resultados de forma más rápida y eficiente; aun así, sigue siendo importante conocer todas las características y diferencias de cada una para ser tomadas en cuenta a la hora de programar. De igual forma, hay que ser consciente de los posibles riesgos de desbordamiento, por lo que es recomendable establecer límites claros y concisos en un algoritmo, sea iterativo o recursivo.

Otro aspecto a resaltar es la utilidad que da el entorno Mars por permitir ver, de forma clara, los pasos por los que se desarrolla un algoritmo, permitiendo una visualización clave para entender el camino por el cual se ejecuta, además de plantear un esbozo de que hace cada instrucción con los respectivos registros