Informe Practica I

Adrian Cordero

8 de febrero de 2026

1. Introducción

El presente informe analiza la implementación de una función de paridad en arquitectura MIPS32, comparando los enfoques recursivo e iterativo. Se detalla el uso de la pila, los riesgos asociados y las diferencias con los modelos teóricos.

2. Implementación de la recursividad y el rol de $sp

La recursividad en MIPS se implementa mediante la gestión manual de la memoria. A diferencia de los lenguajes de alto nivel, el hardware no "recuerda" a dónde debe volver tras una función.

2.1. El Registro $ra (Return Address):

Al usar **jal**, la dirección de retorno se guarda en **$ra**. Si la función se llama a sí misma, ese valor se sobrescribe.

2.2. El Stack Pointer ($sp):

Actúa como un "ancla" en la memoria **RAM**. Para que la recursión funcione, debemos guardar el valor de **$ra** (y cualquier registro $s o $t que queramos preservar) en la pila antes de la siguiente llamada.

2.3. LIFO (Last In, First Out):

Cada llamada reserva espacio, pila crece hacia abajo, **addi $ra, $ra, -4** y cada retorno libera espacio **addi $ra, $ra, 4**

3. Riesgos de desbordamiento y mitigación

Existen dos tipos principales en este contexto:

3.1. Stack Overflow:

Ocurre si **n** es muy grande. Cada llamada recursiva consume 4 bytes (o más si guardas más registros). Si la recursión es muy profunda, la pila choca con los datos globales o el heap.

3.1.1. Mitigación:

Usar implementaciones iterativas o limitar el rango de entrada de **n**.

3.2. Desbordamiento de Enteros:

Si realizas operaciones aritméticas que superen los 32 bits.

3.2.1. Mitigación:

Usar instrucciones como **addu** (unsigned) si no necesitas atrapar excepciones, o validar rangos antes de operar.

4. Diferencias: Iterativo vs. Recursivo

Característica	Implementación Iterativa	Implementación Recursiva
Memoria (Pila)	No usa $sp	Si usa $sp
Registros	$a0, $v1, $t0, $ra	$a0, $v1, $t0, $ra que guarda/restaura contentmente
Velocidad	***O(n)*** solo con saltos j	***O(n)*** con accesos a memoria sw/lw

5. Ejemplos académicos vs. Código operativo

Los libros (como Patterson & Hennessy) suelen mostrar fragmentos aislados para explicar un concepto. Un código operativo en MARS requiere:

5.1. Directivas de datos y texto:

.data y .text.

5.2. Punto de entrada:

La etiqueta **main**: y la llamada al sistema para finalizar el programa:

li $v0, 10

5.3. Preservación de registros:

Los ejemplos de libros a veces omiten el guardado de registros temporales para mayor claridad, pero en un programa real, esto causaría errores de lógica

6. Tutorial paso a paso en MARS

Para probar tu función paridad:

6.1. Ensamblar:

Haz clic en el icono de la llave inglesa y el destornillador. Revisar que no haya errores en la consola inferior.

6.2. Preparar Registros:

En la pestaña Registers", busca $a0 y dale un valor (ej. 5).

6.3. Ejecución Paso a Paso:

Usa el icono de la flecha **verde** con un "**1**".

- Observar como **$sp** disminuye en cada **jal**.
- Mirar la pestaña "Data Segment"para ver cómo se guarda el valor de $ra en la memoria.
- Sigue el flujo hasta que **$a0** llegue a **0** y comience el "desenrollado"de la pila.

6.4. Verificar Resultado:

Al terminar, el resultado final estará en **$v1**.

7. Justificación del enfoque

Para MIPS, el enfoque iterativo es superior.

7.1. Eficiencia:

El procesador MIPS está diseñado para ejecutar instrucciones simples rápido. El acceso a memoria (sw/lw) es costoso en ciclos de reloj.

7.2. Claridad:

Aunque la recursión es elegante matemáticamente, en ensamblador se vuelve difícil de leer por la gestión constante de la pila. La versión iterativa es un bucle limpio de 5 líneas.

8. Análisis y Discusión de los Resultados

- La versión recursiva de paridad(100) hará que el puntero de pila $sp se mueva 100 veces, consumiendo 400 bytes.
- La versión iterativa hará las mismas 100 vueltas pero sin tocar la memoria RAM, manteniendo el rendimiento al máximo.
- Conclusión: La recursividad en ensamblador debe reservarse para algoritmos donde la estructura de datos sea inherentemente recursiva (como árboles), pero para paridad o factoriales, la iteración es la elección técnica correcta.