Informe

Samuel Primera C.I: 31.129.684, Samuel Reyna CI: 30.210.759

8 de Julio del 2025

1. Cómo se implementa la recursividad en MIPS32

Cada llamada recursiva en MIPS32 crea un nuevo marco de pila para aislar variables locales, argumentos y la dirección de retorno:

1. **Al entrar en la función**:
   - Reservar espacio en pila: `addi $sp, $sp, -X`.
   - Guardar registros callee-saved (registros preservados por la función llamada) y `$ra`:

     ```
     sw \$ra, 0(\$sp)
     sw \$s0, 4(\$sp)
     ```

   - Copiar argumentos o variables locales adicionales en la pila.
2. **Al hacer la llamada recursiva**:
   - Cargar nuevos argumentos en `$a0`–`$a3`.
   - Ejecutar `jal NombreFunción`, que guarda en `$ra` la dirección de retorno.
3. **Al regresar de la llamada**:
   - Restaurar registros desde la pila:

     ```
     lw \$ra, 0(\$sp)
     lw \$s0, 4(\$sp)
     ```

   - Liberar espacio en pila: `addi $sp, $sp, X`.
   - Saltar a la dirección de retorno con `jr $ra`.

El registro `$sp` mantiene el puntero de pila, otorgando un área propia a cada nivel de recursión.

2. Riesgos de desbordamiento y cómo mitigarlos

- La recursión profunda puede agotar la memoria de pila (*stack overflow*), corrompiendo datos o provocando fallos.
- Llamadas infinitas o demasiadas llamadas pueden exceder límites del sistema operativo.
- Mitigaciones:
  - Asegurar una condición base clara para limitar la profundidad.
  - Reescribir en forma iterativa cuando convenga.
  - Emplear recursión de cola (tail recursion) para reutilizar el mismo *frame*.
  - Incrementar el tamaño de pila si el entorno lo permite.
  - Incluir chequeos que aborten antes de niveles críticos.

3. Diferencias en uso de memoria y registros: Iterativo vs. Recursivo

Aspecto técnico	**Iterativo**	**Recursivo**
Registros usados	Utiliza registros temporales (`$t0--$t9`) que no necesitan respaldo. Simples de manejar, ideales para cálculos directos sin contaminación de contexto.	Emplea registros de argumento (`$a0--$a3`), de retorno (`$ra`), resultado (`$v0`) y de marco (`$fp`, `$sp`). Requiere respaldo y restauración meticulosa.
Uso de la pila	Pila opcional; sólo se usa si se requiere preservar datos entre iteraciones. Bajo riesgo de desbordamiento.	Cada llamada añade un marco de activación: dirección de retorno, parámetros, locales y registros. Alta presión sobre la pila, riesgo de desbordamiento de datos.
Eficiencia en memoria	Consumo constante y predecible. Apto para sistemas embebidos y estructuras iterativas simples.	Memoria dinámica y creciente con profundidad de llamadas. Puede volverse ineficiente sin optimización.
Control del flujo	Basado en saltos condicionales (`beq`, `bne`, etc.). Estructura clara y fácil de seguir. Ideal para trazado y análisis estático.	Uso de `jal` y `jr` para llamadas anidadas. Control del flujo implícito, menos transparente y más difícil de depurar.
Complejidad del código	Estructura lineal. Código explícito y fácil de mantener, aunque más largo en problemas jerárquicos.	Código conciso pero con mayor abstracción. Ideal para estructuras autosimilares como árboles o algoritmos de retroceso.
Velocidad de ejecución	Más veloz por evitar salto de contexto, respaldo y restauración.	Menor velocidad por sobrecarga de llamadas, manejo de pila y marcos de activación.
Errores comunes	Bucles infinitos si la condición de salida falla. Pérdida de control si se manipulan contadores incorrectamente.	Desbordamiento de datos si falta o no se alcanza la condición base. Pérdida de retorno si `$ra` no se guarda correctamente.
Legibilidad	Muy clara para algoritmos secuenciales simples (e.g., suma acumulativa).	Mejor legibilidad en algoritmos que se describen naturalmente de forma recursiva (e.g., recorrido de árbol, combinatoria).

4. Diferencias entre ejemplos académicos y ejercicios en MIPS32

Mientras los ejercicios del libro solo requieren diseñar funciones específicas, implementarlas en un entorno de ejecución real como `MARS`, que es el que estamos usando, que exige construir un programa completo: incluyeendo declarar variables en la sección `.data`, codificar un punto de entrada `main`, gestionar manualmente los registros según convenciones (preservando `$s0-$s7`), y utilizar *syscalls* para interactuar con el sistema , ya que aquí se debe controlar todo el flujo (desde la inicialización hasta la liberación de recursos) e integrar explícitamente operaciones que en el contexto teórico se asumen abstractas, culminando siempre con una llamada de salida (*syscall* con `$v0=10`) para evitar errores de ejecución.

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

- **Paso 1:** Abrir un archivo .asm en MARS.
- **Paso 2:** Buscar en la barra de herramientas un botón con una llave y un destornillador formando una **X**. Sabrás que es el correcto cuando al pasar el cursor sobre él diga: *"Ensamblar el archivo actual y eliminar los puntos de interrupción"*.
- **Paso 3:** Dos botones a la derecha de este, hay un botón con un círculo verde y un triángulo blanco apuntando a la izquierda, con un pequeño uno abajo. Sabrás que es el correcto cuando al pasar el cursor sobre él diga: *"Ejecutar un paso a la vez"*.
- **Paso 4:** Si necesitas retroceder, a la izquierda del botón anterior hay un botón con un círculo azul y un triángulo blanco apuntando a la derecha, con un pequeño uno abajo. Sabrás que es el correcto cuando al pasar el cursor sobre él diga: *"Regresar al último paso"*.

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

Iterativo

Optamos por el método iterativo ya que es más legible y sencillo de comprender; a diferencia de su versión recursiva, este método no necesita de llamadas recursivas ni el uso de pilas para no perder el valor de los datos. Ambos métodos son similares a la hora de pedir la entrada y las salidas, pero para realizar el cálculo del término n de la secuencia de Fibonacci, el método iterativo es más sencillo y fácil de intuir. En cambio, en el método recursivo es necesario el uso de pilas para guardar la dirección de memoria y otros datos necesarios para realizar las operaciones necesarias; y aunque ambos están en riesgo de desbordamiento aritmético, el método recursivo es propenso a tener desbordamiento en la pila por el simple hecho de utilizarla.

7. Análisis y Discusión de los Resultados

En el caso del Fibonacci iterativo, luego de tomar el n de la entrada verificamos que sea válido mediante un condicional que arrojará un mensaje de error en caso de que no sea válido, antes de entrar a los cálculos de la secuencia Fibonacci es necesario almacenar los primeros dos números de la serie y verificar que el n sea o no uno de esos números, luego entramos a la etiqueta donde se encuentra el ciclo que realiza las operaciones, finalmente imprimimos el valor de la serie y cerramos el programa.

Para el caso del Fibonacci recursivo el procedimiento es similar al iterativo, pero su diferencia radica en el cálculo del n-ésimo término. En este método se realizó un "jal" (jump and link) hacia la función recursiva; seguido se tuvo que cargar una pila con la dirección y los registros necesarios para almacenar los datos. Se realizan los cálculos aritméticos y de asignación necesarios, disminuyendo el n de la entrada hasta o que sea cero. De ahí saltamos a una etiqueta que descargará la pila almacenada hasta que llegó al caso base para que al final retornar a la función principal, imprimir la salida y cerrar el programa.