**Informe**

**1. Recursividad en MIPS32 y el Rol de la Pila ($sp)**

En la arquitectura MIPS32, esto se logra mediante la instrucción jal que salta a una etiqueta y guarda la dirección de la siguiente instrucción (la dirección de retorno) en el registro $ra.

El puntero a pila ($sp) se amplia para guardar todos los datos que se deben preservar en la llamada a la función, como el registro $ra. Esto permite que pueda reanudarse una ejecución anterior sin perder la información de cómo volver al main.

La pila se usa de la siguiente manera:

1. **Reservar Espacio en la Pila:**  
   addi $sp, $sp, -16   
     
   Antes de hacer una llamada recursiva, la función reserva espacio en la pila para guardar los datos que necesita preservar. En este caso, 16 bytes para almacenar cuatro palabras de 4 bytes.
2. **Guardar el Contexto:**  
   sw $a0, 4($sp)   
   sw $ra, 0($sp)   
     
   La función guarda en la pila la dirección de retorno ($ra) y el argumento actual ($a0, que es n). Esto es vital, porque la siguiente llamada a jal fibonacci sobrescribirá el registro $ra. Si no se guardara, la función no sabría a dónde regresar.
3. **Restaurar el Contexto:**  
   lw $ra, 0($sp)   
   lw $t0, 8($sp)   
   lw $t1, 12($sp)   
   addi $sp, $sp, 16   
     
   Una vez que las llamadas recursivas han retornado, la función restaura los valores de la pila a los registros. Libera el espacio de la pila moviendo el puntero ($sp) de vuelta a su posición original. Finalmente, usa jr $ra para volver al punto de ejecución anterior.

**2. Riesgos de Desbordamiento**

* **Desbordamiento de Pila:** Cada llamada recursiva consume espacio (bytes) en la pila. Si el número de entrada n es muy grande, la pila crecerá hasta agotar el espacio de memoria asignado para ella.
  + **Cómo evitarlo:** La mejor forma de evitarlo es no usar una implementación recursiva para número de datos muy grandes, en cambio optar por alternativas iterativas.
* **Desbordamiento de Entero:** La secuencia de Fibonacci crece exponencialmente. Los registros de MIPS32 tienen 32 bits, por lo que el valor máximo que pueden almacenar (para enteros con signo) es 2147483647. Solo el número fib(47) ya supera este límite, produciendo un resultado incorrecto debido al desbordamiento.
  + **Cómo evitarlo:** Se puede implementar una lógica de detección de desbordamiento. Por ejemplo, al sumar dos números positivos (add $v0, $t0, $t1), si el resultado ($v0) es negativo, significa que ha ocurrido un desbordamiento. En ese caso, el programa podría detenerse y mostrar un mensaje de error.

**3. Comparación: Enfoque Iterativo vs. Recursivo**

|  |  |  |
| --- | --- | --- |
| **Característica** | **Enfoque Recursivo (Código provisto)** | **Enfoque Iterativo** |
| **Uso de Memoria** | **Alto.** Consume memoria en la pila con cada llamada. Para fib(n), la profundidad de la pila es proporcional a n. | **Muy Bajo.** Solo necesita un número fijo de registros para almacenar los valores actuales y el contador. No usa la pila para el bucle. |
| **Uso de Registros** | **Intensivo.** Requiere guardar y restaurar registros ($ra, $a0) constantemente en la pila, lo que genera accesos a memoria. | **Eficiente.** Los registros se usan de manera continua dentro de un bucle, sin la sobrecarga de operaciones de guardado y restauración (sw, lw). |
| **Rendimiento** | **Lento.** La sobrecarga de las llamadas a función (jal, jr) y la gestión de la pila lo hacen ineficiente. Además, recalcula los mismos valores múltiples veces (ej. fib(5) calcula fib(3) dos veces). | **Rápido.** Es mucho más veloz porque consiste en un bucle simple sin llamadas a función ni accesos innecesarios a memoria. |
| **Claridad del Código** | **Alta.** Refleja directamente la definición matemática de Fibonacci, lo que puede ser más fácil de entender conceptualmente. | **Moderada.** Requiere entender la lógica de un bucle, pero es el estándar en la práctica por su eficiencia. |

**4. Diferencias entre Ejemplos Académicos y un ejercicios completo y operativo en MIPS32**

El código fibonacci ejercicio.asm es un excelente ejemplo académico, pero no sería adecuado para una aplicación real:

* **Propósito:** El código académico busca enseñar un concepto (recursividad y pila) de la forma más clara posible. Una implementación real prioriza la eficiencia y el manejo de errores.
* **Eficiencia:** En la práctica, nadie usaría una implementación recursiva "ingenua" para Fibonacci debido a su pésimo rendimiento. Se preferiría la versión iterativa.
* **Manejo de Errores:** El código del libro asume una entrada válida (un entero pequeño y no negativo). Un programa real debe validar la entrada: Debería manejar estos casos elegantemente en lugar de fallar o dar un resultado incorrecto.
* **Optimización:** Si se insistiera en la recursividad por alguna razón, una implementación real usaría técnicas como la memoización (guardar los resultados ya calculados en una tabla para no recalcularlos), lo que mejora el rendimiento del enfoque recursivo.

**5. Tutorial Paso a Paso para Ejecutar en MARS**

1. **Abrir MARS:** Inicia el simulador MARS.
2. **Cargar el Archivo:** Ir al menú File > Open... y selecciona tu archivo fibonacci ejercicio.asm.
3. **Ensamblar el Código:** Hacer clic en Run > Assemble en el menú, o presionar la tecla **F3**. El código se compilará a lenguaje máquina. Si no hay errores.
4. **Ejecutar el Programa:** Hacer clic en Run > Go en el menú, o presionar la tecla **F5**.
5. **Interactuar con la Consola:** En la parte inferior de la ventana, la pestaña Run I/O se activará y mostrará el mensaje: Ingresa un número.
6. **Ingresar un Número:** Hacer clic dentro de la ventana de Run I/O, escribir un número entero (por ejemplo, 8) y presionar la tecla Enter.
7. **Ver el Resultado:** El programa calculará el número de Fibonacci y mostrará el resultado en la misma consola (para 8, mostrará 21). El programa finalizará su ejecución.

**6. Justificación del Enfoque Elegido (Recursivo)**

Elegí un enfoque recursivo. Es la manera más directa de traducir la definición matemática F(n) = F(n-1) + F(n-2) a código. Esta elección es excelente para:

* **Demostrar el concepto de recursividad.**
* **Ilustrar la necesidad y el funcionamiento de la pila.**
* **Enseñar cómo se gestionan las llamadas a función (jal) y los retornos (jr $ra).**

Aunque es ineficiente en la práctica, su claridad conceptual lo convierte en un ejercicio clásico en cursos de arquitectura de computadoras y lenguaje ensamblador.

**7. Análisis y Discusión de Resultados**

El programa fibonacci ejercicio.asm cumple su objetivo de calcular correctamente la secuencia de Fibonacci para números enteros pequeños y no negativos. Su implementación es una demostración clara y funcional de la recursividad en MIPS32.

Sin embargo, su análisis revela las importantes **limitaciones** de este enfoque: es lento, consume mucha memoria de pila y es vulnerable a desbordamientos tanto de pila como de enteros.