LABORATORIO 1

Implementación de la Sucesión de Fibonacci en MIPS32

Carlos Sánchez
C.I. V-31030122

Descripción

Este laboratorio consiste en implementar el cálculo del enésimo término de la sucesión de Fibonacci en MIPS32, utilizando dos enfoques: uno iterativo y otro recursivo. Ambos programas reciben como entrada un número entero no negativo n y devuelven como salida el valor de fib(n).

Código Iterativo

```
.text
.globl main

main:
    la $a0, prompt
    li $v0, 4
    syscall

    li $v0, 5
    syscall
    move $t0, $v0

    li $t1, 0
    li $t2, 1
    beq $t0, $zero, print_fib0
    li $t3, 1
    beq $t0, $t3, print_fib1

    li $t3, 2
loop:
    add $t4, $t1, $t2
    move $t1, $t2
    move $t2, $t4
    addi $t3, $t3, 1
```

```
    ble $t3, $t0, loop

    move $a0, $t2
    li $v0, 1
    syscall
    j exit

print_fib0:
    li $a0, 0
    li $v0, 1
    syscall
    j exit

print_fib1:
    li $a0, 1
    li $v0, 1
    syscall

exit:
    li $v0, 10
    syscall

.data
prompt: .asciiz "Ingresa un numero entero no negativo: \n"
```

Código Recursivo

```
.text
.globl main

main:
    la $a0, prompt
    li $v0, 4
    syscall

    li $v0, 5
    syscall
    move $a0, $v0

    jal vfib

    move $a0, $v0
    li $v0, 1
    syscall

    li $v0, 10
```

```
    syscall

vfib:
    addi $t0, $zero, 1
    beq $a0, $zero, fib0
    beq $a0, $t0, fib1
    j fib

fib0:
    li $v0, 0
    jr $ra

fib1:
    li $v0, 1
    jr $ra

fib:
    addi $sp, $sp, -16
    sw $ra, 0($sp)
    sw $a0, 4($sp)

    addi $a0, $a0, -1
    jal vfib
    sw $v0, 8($sp)

    lw $a0, 4($sp)
    addi $a0, $a0, -2
    jal vfib
    sw $v0, 12($sp)

    lw $ra, 0($sp)
    lw $t0, 8($sp)
    lw $t1, 12($sp)
    addi $sp, $sp, 16
    add $v0, $t0, $t1

    jr $ra

.data
prompt: .ascii "Ingresa un numero entero no negativo: \n"
```

Preguntas y Respuestas

1. ¿Cómo se implementa la recursividad en MIPS32 y qué papel cumple la pila `$sp`?

La recursividad se implementa mediante llamadas a funciones con `jal`, y cada llamada guarda su contexto (como `$ra` y `$a0`) en la pila usando `$sp`. Esto permite que cada llamada tenga su propio entorno de ejecución sin interferencias.

2. ¿Qué riesgos de desbordamiento existen y cómo mitigarlos?

Existen dos riesgos:

- **Desbordamiento de pila:** muchas llamadas recursivas pueden agotar la memoria de pila.
- **Desbordamiento de enteros:** los valores de Fibonacci crecen rápidamente y pueden exceder los 32 bits.

Se mitigan limitando el valor de entrada y prefiriendo la versión iterativa.

3. Diferencias entre implementación iterativa y recursiva

- La versión iterativa usa menos memoria y es más rápida.
- La versión recursiva es más elegante pero consume más pila y es más lenta.

4. Tutorial paso a paso en MARS

1. Abrir MARS y cargar el archivo `.asm`.
2. Ensamblar con `Assemble`.
3. Usar `Step` para ejecutar instrucción por instrucción.
4. Observar los registros y la pila en cada paso.

5. Justificación del enfoque

El enfoque iterativo es más eficiente y seguro en MIPS32, ya que evita el uso excesivo de la pila y es más directo. La recursividad es útil para aprendizaje, pero menos práctica en arquitecturas con recursos limitados.