



IIC2343 - Arquitectura de Computadores (II/2025)

Actividad de programación

Sección 2 - Pauta de evaluación

Pregunta 1: Explique el código (3 ptos.)

En el siguiente fragmento de código se realiza el llamado de una subrutina `func_n_m`:

```
.data
n: .word 37
m: .word 5
r: .word 0

.text
main:
    addi sp, sp, -16
    sw ra, 12(sp)

    la t0, n
    lw a0, 0(t0)
    la t1, m
    lw a1, 0(t1)

    jal ra, func_n_m

    la t2, r
    sw a0, 0(t2)

    lw ra, 12(sp)
    addi sp, sp, 16
    addi a7, zero, 10
    ecall

func_n_m:
    beq a0, zero, ccc

    addi sp, sp, -16
    sw ra, 12(sp)
    sw a0, 8(sp)

    addi a0, a0, -1
    jal ra, func_n_m

    lw t0, 8(sp)
    lw ra, 12(sp)
    addi sp, sp, 16

    add a0, a0, a1
    jalr zero, 0(ra)

ccc:
    addi a0, zero, 0
    jalr zero, 0(ra)
```

Este fragmento representa el cómputo de una función $f(n, m)$. A partir de este:

1. (1.5 ptos.) Indique, con argumentos y en términos de n y m , lo que retorna la función $f(n, m)$. Por ejemplo, $f(n, m) = n + m$. Se otorgan **0.75 ptos.** por la correctitud de la descripción del retorno y **0.75 ptos.** por justificación.

Solución: La función implementa la multiplicación mediante sumas sucesivas, por lo que:

$$f(n, m) = n \times m$$

En el código, `a0` contiene el argumento n y actúa como contador decreciente en cada llamado recursivo. El caso base ocurre cuando $a0 = 0$, retornando 0. En cada retorno desde la recursión, se suma `a1` (que contiene m) al valor acumulado en `a0`, de modo que se realizan n sumas del valor m . El resultado final ($n \times m$) queda almacenado en `a0`, y luego se guarda en `r`.

2. (1.5 ptos.) Indique, con argumentos, si el fragmento anterior respeta o no la convención de llamadas de RISC-V. Se otorgan **0.75 ptos.** si indica de forma correcta si se respeta o no la convención y **0.75 ptos.** por entregar una justificación válida respecto a su respuesta.

Solución: El fragmento anterior **NO respeta** la convención de llamadas de RISC-V.

Aunque el código ejecuta correctamente la multiplicación y maneja el `stack` sin errores de ejecución, desde el punto de vista de la convención formal no cumple con las reglas establecidas. En particular:

- Los registros `a0` y `a1` son *caller-saved*. En el programa principal (`main`) se realiza el llamado a `func_n_m` sin respaldar previamente ninguno de estos registros, a pesar de que ambos pueden ser modificados por la subrutina.
- Dentro de la subrutina `func_n_m`, el registro `a1` tampoco es respaldado antes del llamado recursivo, incumpliendo nuevamente el requisito de *caller-saved*, ya que la función se llama a sí misma y puede alterar sus propios argumentos.
- El registro `t0`, que es de tipo *caller-saved*, se utiliza dentro de la función después del retorno recursivo sin haber sido respaldado antes del llamado, lo que también infringe la convención.

Pregunta 2: Elabore el código (3 ptos.)

Elabore utilizando el Assembly RISC-V un programa que, dado un arreglo **arr** de largo **len** de enteros positivos, realice lo siguiente:

Para cada posición **i** del arreglo:

- Si **i** != **len** - 1, que tome **arr[i]** como **base** y **arr[i+1]** como **exponente**, y reemplace **arr[i+1]** por **arr[i]^{arr[i+1]}**.
- Si **i** == **len** - 1, *i.e.* si se trata del último elemento, se reemplaza por su valor al cuadrado: **arr[i] = arr[i]²**.

Al finalizar, el programa debe calcular la **suma de todos los valores del arreglo actualizado** y guardar el resultado en la variable **compressed**.

Solución: En la siguiente plana se presenta una solución que implementa correctamente el comportamiento solicitado.

El programa recorre el arreglo **arr** desde el índice **i** = 0 hasta **i** = **len** - 1 realizando los siguientes pasos:

- Carga el valor actual **arr[i]** y determina si se trata del último elemento del arreglo.
- Si **i** < **len** - 1, guarda **arr[i+1]** y calcula **arr[i]^{arr[i+1]}** invocando la subrutina **power**. El resultado reemplaza el valor original en **arr[i+1]**.
- Si **i** == **len** - 1, eleva el último elemento al cuadrado (**arr[i]²**) usando la misma subrutina.
- Finalizado el programa efectúa una segunda iteración para sumar todos los valores de **arr** y guarda el total en la variable **compressed**.

La subrutina **power** implementa la operación de elevar de forma iterativa mediante multiplicaciones sucesivas:

- Inicializa el acumulador **a0** = 1.
- Mientras el exponente **a2** sea distinto de cero, multiplica **a0** = **a0 * a1** y decrementa el exponente.
- Retorna al llamador mediante **jalr zero, 0(ra)**.

```

.data
arr:      .word 2,2,1      # Arreglo
len:      .word 3          # Largo del arreglo
compressed: .word -1        # Resultado

.text
main:
la t0, arr
la t1, len
lw t1, 0(t1)

addi t2, zero, 0           # i = 0

loop:
bge t2, t1, end_loop      # if i >= len to end
slli t4, t2, 2              # t4 = i * 4
add t4, t0, t4
lw t5, 0(t4)                # t5 = arr[i]
addi t6, t1, -1             # t6 = len - 1
beq t2, t6, last_element   # if i == len-1 to last_element
lw a2, 4(t4)                # a2 = arr[i+1]
add a1, t5, zero             # a1 = arr[i]
jal ra, power               # call power
sw a0, 4(t4)                # arr[i+1] = result
beq zero, zero, continue_loop
last_element:
add a1, t5, zero             # a1 = arr[i]
addi a2, zero, 2              # a2 = 2
jal ra, power
sw a0, 0(t4)
continue_loop:
addi t2, t2, 1                # i++
beq zero, zero, loop
end_loop:
la t0, arr
addi t2, zero, 0
addi t3, zero, 0
sum_loop:
bge t2, t1, end_sum
slli t4, t2, 2
add t4, t0, t4
lw t5, 0(t4)
add t3, t3, t5
addi t2, t2, 1
beq zero, zero, sum_loop

end_sum:
la t6, compressed
sw t3, 0(t6)
addi a7, zero, 10
ecall

power:
addi a0, zero, 1
beq a2, zero, power_end

power_loop:
beq a2, zero, power_end
mul a0, a0, a1
addi a2, a2, -1
beq zero, zero, power_loop

power_end:
jalr zero, 0(ra)

```

El puntaje se distribuye de la siguiente forma:

- **1.0 pto.** — **Estructura general del algoritmo:** recorrido secuencial del arreglo con buen flujo de control y manipulación correcta de índices.
- **1.0 pto.** — **Cálculo de potencia:** implementación iterativa correcta de potencia entre dos números.
- **0.5 ptos.** — **Tratamiento del último elemento:** identificación de `i == len - 1` y aplicación del cuadrado.
- **0.5 ptos.** — **Suma final del arreglo:** recorrido completo y almacenamiento del resultado en `compressed`.

Si el código muestra la estructura y la intención del algoritmo (recorrido, uso de subrutina de potencia, o suma parcial), pero presenta errores menores de control o de saltos, se podrá otorgar **hasta 0.75 ptos.** por aproximación.