DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (II/2025)

Actividad de programación formativa

Sección 3 - 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 7
    m:      .word 5
    res:    .word 0

.text
    main:
        la t0, n
        la t1, m
        la t2, res
        lw a0, 0(t0)
        lw a1, 0(t1)
        addi sp, sp, -4
        sw ra, 0(sp)
        jal ra, func_n_m
        lw ra, 0(sp)
        addi sp, sp, 4
        sw a0, 0(t2)
        addi a7, zero, 10
        ecall
    func_n_m:
        addi sp, sp, -12
        sw ra, 0(sp)
        sw a0, 4(sp)
        sw t2, 8(sp)
        add t2, zero, zero
        beq a0, t2, base_end
        beq a1, t2, base_end
        addi a1, a1, -1
        jal ra, func_n_m
        lw t0, 4(sp)
        add a0, t0, a0
        jal zero, end
        base_end:
            add a0, zero, zero
        end:
            lw ra, 0(sp)
            lw t2, 8(sp)
            addi sp, sp, 12
            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 computa el producto de m con n, *i.e.* $f(n, m) = n \times m$. Esto se evidencia en el hecho de que el valor de retorno, almacenado en el registro `a0`, es igual a $n + f(n, m - 1)$, con caso base $f(n, 0) = n$. De esta forma, se tiene que:

   $$f(n, m) = n + f(n, m - 1) = n + n + f(n, m - 2) = \cdots = n + n + n + \cdots + n + 0 = n \times m$$

   (se puede corroborar que el término n se suma m veces). No es necesario que se haga un detalle completo del código, pero sí que se demuestre que se entiende el cómputo recursivo de la potencia de x a la n.

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 ptos.** si su respuesta es incorrecta. Por otra parte, se otorgan **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. Si bien se cumplen los siguientes criterios:

   - Se respalda `ra` antes de cada llamado.
   - Se usan los registros `a0` y `a1` para los argumentos de `func_n_m`, y `a0` para su retorno.
   - Se respalda el registro `a0` antes de cada llamado recursivo, considerando su sobreescritura; su uso posterior al llamado; y el hecho de ser *caller-saved*.

   Existe una falta importante: El registro `t2`, utilizado tanto dentro como fuera de la subrutina `func_n_m`, **se respalda dentro de ella**. Esta es una falta a la convención dado que los registros `t*` son *caller-saved* y no *callee-saved*, es decir, se debió haber respaldado fuera de la subrutina.

Pregunta 2: Elabore el código (3 ptos.)

Elabore, utilizando el Assembly RISC-V, un programa que a partir de un arreglo `arr` de largo `len`, determine si este posee un par de números **en posiciones distintas** tal que su suma sea igual a un valor `pair_sum`. Si se encuentra un par de números que cumple lo pedido, se deben guardar los índices del par en las variables `x_pair_sum` e `y_pair_sum`; en otro caso, sus valores deben ser iguales a -1. Si existe más de un par de valores que cumple lo pedido, puede escoger arbitrariamente los primeros que encuentre.

A continuación, tres ejemplos:

$$\begin{aligned}
\texttt{arr} = [1,3,5,7,-5] \quad \texttt{pair_sum} = 10 &\rightarrow (\texttt{x_pair_sum}, \texttt{y_pair_sum}) = \quad (1,3) \\
\texttt{arr} = [1,3,5,7,-5] \quad \texttt{pair_sum} = 11 &\rightarrow (\texttt{x_pair_sum}, \texttt{y_pair_sum}) = \quad (-1,-1) \\
\texttt{arr} = [1,3,5,7,-5] \quad \texttt{pair_sum} = 14 &\rightarrow (\texttt{x_pair_sum}, \texttt{y_pair_sum}) = \quad (-1,-1)
\end{aligned}$$

Puede utilizar el siguiente fragmento de código como base:

```
.data
  arr:          .word 1, 3, 5, 7, -5
  len:          .word 5
  pair_sum:     .word 10
  x_pair_sum:   .word -1
  y_pair_sum:   .word -1
.text
  # Su código aquí.
```

La asignación de puntaje se distribuirá de la siguiente forma:

- **1 pto.** por resolver correctamente el caso donde no existe un par. Se descuentan **0.5 ptos.** si existe como máximo un error de implementación y no se asigna puntaje si existe más de uno.
- **2 ptos.** por resolver correctamente el caso donde sí existe al menos un par. Se descuenta **1 pto.** si existe como máximo un error de implementación y no se asigna puntaje si existe más de uno.

IMPORTANTE: No es necesario que respete la convención en este ejercicio.

Solución: En la siguiente plana, se muestra una solución que utiliza dos índices `i`, `j` para recorrer el arreglo y revisar todos los pares de valores.
IMPORTANTE: Si bien no figura en el enunciado, durante la actividad se señala que se puede agregar la variable `len` para guardar el largo del arreglo.

```
.data
  arr:        .word 1, 3, 5, 7, -5
  len:        .word 5
  pair_sum:   .word 10
  x_pair_sum: .word -1
  y_pair_sum: .word -1
.text
  la s0, arr                      # Dirección inicial del arreglo.
  addi s1, zero, 4                # Constante para multiplicar índices por 4 para obtener direcciones.
  lw s2, len                      # Largo del arreglo.
  addi s3, s2, -1                 # len - 1 para terminar antes el loop de i.
  lw s4, pair_sum                 # Suma a buscar.
  la s5, x_pair_sum               # Dirección para almacenar el índice del primer número del par.
  la s6, y_pair_sum               # Dirección para almacenar el índice del segundo número del par.
  addi t0, zero, 0                # Índice i
  find_pair_sum:
    i_loop:                       # Iteración respecto al índice i
      addi t1, t0, 1              # j = i + 1, no vemos los elementos hacia atrás, ya se revisaron.
      j_loop:
        mul t2, t0, s1            # t2 = 4 * i
        mul t3, t1, s1            # t3 = 4 * j
        add t2, s0, t2            # t2 = dir(arr) + 4 * i
        add t3, s0, t3            # t3 = dir(arr) + 4 * j
        lw t2, 0(t2)              # t2 = arr[i]
        lw t3, 0(t3)              # t3 = arr[j]
        add t4, t2, t3            # t4 = arr[i] + arr[j]
        beq t4, s4, pair_sum_found  # arr[i] + arr[j] == pair_sum
      j_loop_continue:
        addi t1, t1, 1
        blt t1, s2, j_loop        # if j >= n, end j_loop
    i_loop_continue:
      addi t0, t0, 1
      bge t0, s3, end             # if i >= n - 1, end i_loop
      jal zero, i_loop
    pair_sum_found:
      sw t0, 0(s5)                # x_pair_sum = i
      sw t1, 0(s6)                # y_pair_sum = j
  end:
    addi a7, zero, 10
    ecall
```

La asignación de puntaje se distribuirá de la siguiente forma:

- **1 pto.** por detectar correctamente los casos donde no existe un par. Solo se asigna puntaje si **se comparan los pares de valores del arreglo**. En este criterio no se asigna puntaje parcial.
- **2 ptos.** por obtener correctamente las coordenadas del par de valores que cumplen con el valor de la suma. Se descuentan **0.5 ptos.** si se almacena el valor de los números o de sus direcciones de memoria en vez de sus índices en el arreglo. Si existe más de un error de implementación, no se asigna puntaje.