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

IIC2343 - Arquitectura de Computadores (II/2025)

Actividad de programación

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_m_n`:

```
.data
    m:      .word 15
    n:      .word 6
    res_1:  .word -1
    res_2:  .word -1
.text
    main:
        la t0, m
        la t1, n
        la s0, res_1
        la s1, res_2
        lw a0, 0(t0)
        lw a1, 0(t1)
        addi sp, sp, -8
        sw ra, 0(sp)
        sw s0, 4(sp)
        jal ra, func_m_n
        lw ra, 0(sp)
        lw s0, 4(sp)
        addi sp, sp, 8
        sw a0, 0(s0)
        sw a1, 0(s1)
        addi a7, zero, 10
        ecall
    func_m_n:
        addi sp, sp, -4
        sw ra, 0(sp)
        blt a0, a1, base_end
        sub s0, a0, a1
        add a0, s0, zero
        jal ra, func_m_n
        addi a0, a0, 1
        jal zero, end
        base_end:
            add a1, a0, zero
            add a0, zero, zero
        end:
            lw ra, 0(sp)
            addi sp, sp, 4
            jalr zero, 0(ra)
```

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

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

   Solución: La función calcula la división entera y el resto entre m y n, *i.e.* $f(m, n) = (m//n, m\,\%n)$. Esto se evidencia en el hecho de que el primer valor de retorno, almacenado en el registro `a0`, es igual a $1 + f(m - n, n)$, con caso base $f(m - k \times n, n) = 0, k = \lfloor \frac{m}{n} \rfloor$. Esto hará que se llame a la función $\lfloor \frac{m}{n} \rfloor + 1$ veces, siendo el último retorno igual a cero. Por ende, al hacer la suma, se tendrá que se suma una unidad $\lfloor \frac{m}{n} \rfloor$ veces, es decir, se computará la división entera. Por otro lado, el segundo valor de retorno, almacenado en el registro `a1`, se computa solo en el caso base y corresponde al valor $m - k \times n$ con el k antes señalado, por lo que matemáticamente corresponde al resto de la división. De esta forma, se tiene que:

   $$\begin{aligned} f(m, n) &= (1 + f(m - n, n), m\,\%n) \\ &= (1 + 1 + f(m - 2 \times n, n, m\,\%n) \\ &= \cdots \\ &= (1 + 1 + 1 + \cdots + 1 + 0, m\,\%n) \\ &= (m//n, m\,\%n) \end{aligned}$$

   No es necesario que se haga un detalle completo del código, pero sí que se demuestre que se entiende el cómputo recursivo de la división entera.

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` tanto para los argumentos de `func_m_n` como para sus dos valores de retorno.

   Existe una falta importante: El registro `s0`, utilizado tanto dentro como fuera de la subrutina `func_m_n`, **se respalda fuera de ella**. Esta es una falta a la convención dado que los registros `s*` son *callee-saved* y no *caller-saved*, es decir, se debió haber respaldado dentro de la subrutina.

Pregunta 2: Elabore el código (3 ptos.)

Elabore, utilizando el Assembly RISC-V, un programa que, dado un arreglo `arr` de largo `len` cuyos elementos son enteros de 32 bits (`.word`), determine si este es un "palíndromo" (1) o no (0), es decir, que si se invierte el orden de sus elementos, el arreglo no cambia. El resultado se debe computar en la variable `is_palindrome`.

A continuación, tres ejemplos:

$$\begin{aligned} \texttt{arr} = [1,3,3,1] &\rightarrow \texttt{is\_palindrome} = 1 \\ \texttt{arr} = [1,2,3,2,1] &\rightarrow \texttt{is\_palindrome} = 1 \\ \texttt{arr} = [1,2,3,4,5,6] &\rightarrow \texttt{is\_palindrome} = 0 \end{aligned}$$

Puede utilizar el siguiente fragmento de código como base:

```
.data
    arr:            .word 1,3,3,1  # Arreglo
    len:            .word 4        # Largo del arreglo
    is_palindrome:  .word -1       # Es palíndromo => TRUE = 1, FALSE = 0
.text
  # Su código aquí
```

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

Solución: En la siguiente plana, se muestra una solución que utiliza dos índices `i`, `j` tales que inicialmente `i = 0` y `j = len - 1`. Luego, se realiza el siguiente procedimiento iterativo:

- Si `i >= j`, el programa termina y el arreglo se considera palíndromo.
- Si no ocurre lo anterior, se revisa si `arr[i] == arr[j]`. Si no se cumple la condición, el programa termina y el arreglo no se considera palíndromo.
- Si la condición se cumple, se repite la iteración con nuevos índices: `i += 1`, `j -= 1`.

IMPORTANTE: En el programa base entregado se tenía en la definición de variables `is_palindrome: .word 0`. Se solicita durante la actividad que se corrija a -1, ya que el programa debe configurar el valor en 0 en caso de que el arreglo no sea palíndromo.

```
.data
    arr:             .word 1,2,5,3,2,1  # Arreglo
    len:             .word 6            # Largo del arreglo
    is_palindrome:   .word -1           # Es palíndromo => TRUE = 1, FALSE = 0
.text
  la s0, arr              # Dirección de memoria del arreglo.
  la s1, is_palindrome    # Dirección de memoria del resultado.
  addi s2, zero, 4        # Constante para computar dirección de memoria.
  addi s3, zero, 1        # Constante para cargar un 1 en is_palindrome.
  addi t0, zero, 0        # Índice i.
  lw t1, len
  addi t1, t1, -1         # Índice j.
  loop:
    bge t0, t1, is_true   # Si i >= j, recorrimos todo el arreglo y el arreglo es palíndromo.
    mul a0, t0, s2        # a0 = 4*i
    add a0, a0, s0        # a0 = dirección de memoria arr[i]
    lw a0, 0(a0)          # a0 = arr[i]
    mul a1, t1, s2        # a1 = 4*j
    add a1, a1, s0        # a1 = dirección de memoria arr[j]
    lw a1, 0(a1)          # a1 = arr[j]
    bne a0, a1, is_false  # Si a[i] != a[j], el arreglo no es palíndromo
    addi t0, t0, 1        # i += 1
    addi t1, t1, -1       # j -= 1
    jal zero, loop
  is_true:
    sw s3, 0(s1)          # is_palindrome = 1
    jal zero, end
  is_false:               # is_palindrome = 0
    sw zero, 0(s3)
  end:
    addi a7, zero, 10
    ecall
```

El puntaje se distribuye de la siguiente forma:

- **0.75 ptos.** si el programa computa correctamente `is_palindrome = 1` para arreglos de largo par.
- **0.75 ptos.** si el programa computa correctamente `is_palindrome = 1` para arreglos de largo impar.
- **0.75 ptos.** si el programa computa correctamente `is_palindrome = 0` para arreglos de largo par.
- **0.75 ptos.** si el programa computa correctamente `is_palindrome = 0` para arreglos de largo impar.

En caso de que no se llegue a la solución correcta en ningún caso, o bien que el programa se caiga, se otorgarán **0.75 ptos.** si se evidencia a nivel de código un intento de algoritmo cercano a lo esperado.