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

IIC2343 - Arquitectura de Computadores (II/2023)

Actividad práctica 2

Sección 3 - Pauta de evaluación

Pregunta 1: Explique la convención del código (1 ptos.)

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

```
.data
  N:          .word 17
  N_is_prime: .word -1
.text
  main:
    lw a0, N
    li s0, 1
    la s1, N_is_prime
    call check_is_prime
    sw a0, 0(s1)
    li a7, 10
    ecall
  check_is_prime:
    ble a0, s0, is_not_prime
    addi t1, a0, -1
    li t0, 2
    division_loop:
      rem t2, a0, t0
      beqz t2, is_not_prime
      addi t0, t0, 1
      ble t0, t1, division_loop
    is_prime:
      li a0, 1
      j end
    is_not_prime:
      li a0, 0
    end:
      ret
```

Indique, con argumentos, si el fragmento anterior respeta o no la convención de llamadas de RISC-V. Se otorgan **0.5** 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 **0.5 ptos.** por entregar una justificación válida respecto a su respuesta.

Solución: El fragmento anterior **no respeta la convención de llamadas de RISC-V** dado que no se respalda el registro `ra` antes de llevar a cabo el llamado de una subrutina. Se otorgan **0.5 ptos.** por señalar que la convención no se respeta y **0.5 ptos.** por entregar un argumento válido respecto al uso de los registros, independiente de la correctitud del punto anterior.

Pregunta 2: Arregle el código (2 ptos.)

El siguiente programa, desarrollado en el Assembly RISC-V, **debería computar de forma recursiva la potencia de `x_value` a la `n` y guardar el resultado en `x_pow_n`**:

```
.data
  x_value: .word 7
  n:       .word 3
  x_pow_n: .word 0
.text
  main:
    lw a0, x_value
    lw a1, n
    la s0, x_pow_n
    li s1, 1
    addi sp, sp, -4
    sw ra, 0(sp)
    call pow
    lw ra, 0(sp)
    sw a0, 0(s0)
    li a7, 10
    ecall
  pow:
    beq a1, s1, end
    addi sp, sp, -8
    sw ra, 0(sp)
    sw a0, 4(sp)
    addi a1, a1, -1
    call pow
    lw ra, 0(sp)
    lw t0, 4(sp)
    mul a0, a0, t0
    end:
      ret
```

Sin embargo, no lo hace correctamente. Si compila y ejecuta el código, se dará cuenta que no entrega el resultado esperado. Busque el error y, una vez encontrado, arréglelo para que el fragmento anterior funcione correctamente. Suba el código completo como respuesta. Se otorga **1 pto.** del total si encuentra el error y trata de arreglarlo sin éxito **solo si comenta correctamente en su respuesta el motivo de fallo**. No se otorga puntaje parcial si el programa no compila o se sube sin arreglo ni comentario alguno.

Solución: El problema radica en que **no se restaura el valor de sp posterior a la recuperación de los registros respaldados**. Basta con restaurarlo después de obtener del *stack* los registros respaldados, como se muestra a continuación:

```
.data
  x_value: .word 7
  n:       .word 3
  x_pow_n: .word 0
.text
  main:
    lw a0, x_value
    lw a1, n
    la s0, x_pow_n
    li s1, 1
    addi sp, sp, -4
    sw ra, 0(sp)
    call pow
    lw ra, 0(sp)
    addi sp, sp, 4
    sw a0, 0(s0)
    li a7, 10
    ecall
  pow:
    beq a1, s1, end
    addi sp, sp, -8
    sw ra, 0(sp)
    sw a0, 4(sp)
    addi a1, a1, -1
    call pow
    lw ra, 0(sp)
    lw t0, 4(sp)
    addi sp, sp, 8
    mul a0, a0, t0
    end:
      ret
```

El puntaje se asigna según lo descrito en el enunciado. No hay descuento si hay más modificaciones de las necesarias, pero sí se descuenta **1 pto.** si el valor final no se almacena de forma correcta en la variable `x_pow_n`.

Pregunta 3: 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:

$$\texttt{arr} = [1, 3, 5, 7, -5] \quad \texttt{pair\_sum} = 10 \rightarrow (\texttt{x\_pair\_sum}, \texttt{y\_pair\_sum}) = \quad (1, 3)$$
$$\texttt{arr} = [1, 3, 5, 7, -5] \quad \texttt{pair\_sum} = 11 \rightarrow (\texttt{x\_pair\_sum}, \texttt{y\_pair\_sum}) = \quad (-1, -1)$$
$$\texttt{arr} = [1, 3, 5, 7, -5] \quad \texttt{pair\_sum} = 14 \rightarrow (\texttt{x\_pair\_sum}, \texttt{y\_pair\_sum}) = \quad (-1, -1)$$

Puede utilizar el siguiente fragmento de código como base:

```
.data
  arr:         .word 1, 3, 5, 7, -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.
  li s1, 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.
  li t0, 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
      j i_loop
    pair_sum_found:
      sw t0, 0(s5)                  # x_pair_sum = i
      sw t1, 0(s6)                  # y_pair_sum = j
  end:
    li a7, 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.