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

IIC2343 - Arquitectura de Computadores (II/2023)

Actividad práctica 2

Sección 4 - 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                        # Carga el valor de N en a0
    addi t0, zero, 1                # Carga el literal 1 en t0
    la t1, N_is_prime               # Carga la dirección de N_is_prime en t1
    addi sp, sp, -4
    sw ra, 0(sp)
    jal ra, check_is_prime          # Salto a check_is_prime y guarda retorno en ra
    lw ra, 0(sp)
    addi sp, sp, 4
    la t1, N_is_prime               # Carga la dirección de N_is_prime en t1
    sw a0, 0(t1)
    addi a7, zero, 10               # Carga el literal 10 en a7
    ecall                           # Exit. Solo termina el programa
  check_is_prime:
    ble a0, t0, is_not_prime        # Salto condicional a is_not_prime si a0 <= t0
    addi t1, a0, -1
    addi t0, zero, 2                # Carga el literal 2 en t0
    division_loop:
      rem t2, a0, t0
      beq t2, zero, is_not_prime    # Salto condicional a is_not_prime si t2 == 0
      addi t0, t0, 1
      ble t0, t1, division_loop     # Salto condicional a division_loop si t0 <= t1
    is_prime:
      addi a0, zero, 1              # Carga el literal 1 en a0
      beq zero, zero, end           # Salto incondicional a end
    is_not_prime:
      addi a0, zero, 0              # Carga el literal 0 en a0
    end:
      jalr zero, 0(ra)              # Retorno usando dirección almacenada en ra
```

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 el valor del registro `t1` es sobreescrito por `check_is_prime` y no se respalda a pesar de ser *caller saved*. 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      # Carga el valor de x_value en a0
    lw a1, n            # Carga el valor de n en a1
    la s0, x_pow_n      # Carga la dirección de x_pow_n en s0
    addi s1, zero, 1    # Carga el literal 1 en s1
    addi sp, sp, -1
    sw ra, 0(sp)
    jal ra, pow         # Salto a pow y guarda retorno en ra
    lw ra, 0(sp)
    addi sp, sp, 1
    sw a0, 0(s0)
    addi a7, zero, 10   # Carga el literal 10 en a7
    ecall               # Exit. Solo termina el programa
  pow:
    beq a1, s1, end
    addi sp, sp, -2
    sw ra, 0(sp)
    sw a0, 1(sp)
    addi a1, a1, -1
    jal ra, pow
    lw ra, 0(sp)
    lw t0, 1(sp)
    addi sp, sp, 2
    mul a0, a0, t0
    end:
      jalr zero, 0(ra)  # Retorno usando dirección almacenada en ra
```

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 **se reserva una dirección por registro en vez de 4**. Basta con desplazar `sp` en 4 unidades por registro respaldado y acceder a estos a partir de *offsets* que sean múltiplos de 4, 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
    addi s1, zero, 1
    addi sp, sp, -4
    sw ra, 0(sp)
    jal ra, pow
    lw ra, 0(sp)
    addi sp, sp, 4
    sw a0, 0(s0)
    addi a7, zero, 10
    ecall
  pow:
    beq a1, s1, end
    addi sp, sp, -8
    sw ra, 0(sp)
    sw a0, 4(sp)
    addi a1, a1, -1
    jal ra, pow
    lw ra, 0(sp)
    lw t0, 4(sp)
    addi sp, sp, 8
    mul a0, a0, t0
    end:
      jalr zero, 0(ra)
```

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 **calcule el área del triángulo que se forma a partir de tres puntos en un plano de dos dimensiones**. Las coordenadas de los puntos se guardarán individualmente en la memoria, como se muestra en el fragmento de código a continuación:

```
# Calcular el área de un triángulo que tiene un segmento paralelo al eje x
.data
  area:   .word 0
  base:   .word 0
  altura: .word 0
  P1:     .word 6, 1   # Coordenadas (X,Y) del punto P1
  P2:     .word 8, 10  # Coordenadas (X,Y) del punto P2
  P3:     .word 1, 10  # Coordenadas (X,Y) del punto P3
.text
  # Su código aquí
```

Puede asumir los siguientes supuestos:

- Los tres puntos **siempre** formarán un triángulo válido.
- **Siempre** habrá un lado del triángulo paralelo al eje x.
- Los puntos `P1`, `P2` y `P3` **no siguen un orden predeterminado**. Es su tarea encontrar los puntos que forman la base y altura del triángulo.

La asignación de puntaje se distribuirá de la siguiente forma:

- **1 pto.** por calcular correctamente la base. Se descuentan **0.5 ptos.** si existe como máximo un error de implementación y no se asigna puntaje si existe más de uno.
- **1 pto.** por calcular correctamente la altura. Se descuentan **0.5 ptos.** si existe como máximo un error de implementación y no se asigna puntaje si existe más de uno.
- **1 pto.** por calcular correctamente el área del triángulo. Se descuentan **0.5 ptos.** 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 busca los dos puntos correspondientes a la base para luego calcular la altura.

```
.data
  area:   .word 0
  base:   .word 0
  altura: .word 0
  P1:     .word 6, 1            # Coordenadas (X,Y) del punto P1
  P2:     .word 8, 10           # Coordenadas (X,Y) del punto P2
  P3:     .word 1, 10           # Coordenadas (X,Y) del punto P3
.text
  la s0, area                   # Dirección de la variable area.
  la s1, base                   # Dirección de la variable base.
  la s2, altura                 # Dirección de la variable altura.
  la s3, P1                     # Dirección del elemento X de P1.
  addi s4, s3, 4                # Dirección del elemento Y de P1.
  la s5, P2                     # Dirección del elemento X de P2.
  addi s6, s5, 4                # Dirección del elemento Y de P2.
  la s7, P3                     # Dirección del elemento X de P3.
  addi s8, s7, 4                # Dirección del elemento Y de P3.
  lw s3, 0(s3)                  # s3 = P1.x
  lw s4, 0(s4)                  # s4 = P1.y
  lw s5, 0(s5)                  # s5 = P2.x
  lw s6, 0(s6)                  # s6 = P2.y
  lw s7, 0(s7)                  # s7 = P3.x
  lw s8, 0(s8)                  # s8 = P3.y
  addi s9, zero, 2              # s9 = 2 = divisor para calcular el área.
  beq s4, s8, base_P1_P3        # P1.y == P3.y => base = P1,P3
  beq s6, s8, base_P2_P3        # P2.Y == P3.Y => base = P2,P3
  base_P1_P2:                   # else: base = P1,P2
    add t0, zero, s3            # x1 = P1.x
    add t1, zero, s5            # x2 = P2.x
    add t2, zero, s4            # y1 = P1.y
    add t3, zero, s8            # y2 = P3.y
    jal zero, continue
  base_P1_P3:
    add t0, zero, s3            # x1 = P1.x
    add t1, zero, s7            # x2 = P3.x
    add t2, zero, s4            # y1 = P1.y
    add t3, zero, s6            # y2 = P2.y
    jal zero, continue
  base_P2_P3:
    add t0, zero, s5            # x1 = P2.x
    add t1, zero, s7            # x2 = P3.x
    add t2, zero, s6            # y1 = P2.y
    add t3, zero, s4            # y2 = P1.y
  continue:
    get_base:
      sub t4, t1, t0            # t4 = t1 - t0 = x2 - x1
      blt zero, t4, get_height
      sub t4, zero, t4          # Si t4 < 0, obtenemos su inverso aditivo.
    get_height:
      sub t5, t3, t2            # t5 = t3 - t2 = y2 - y1
      blt zero, t5, get_area
      sub t5, zero, t5          # Si t5 < 0, obtenemos su inverso aditivo.
    get_area:
      mul t6, t4, t5            # t6 = base * altura
      div t6, t6, s9            # t6 = (base * altura) // 2
  end:
    sw t4, 0(s1)                # base = t4
    sw t5, 0(s2)                # altura = t5
    sw t6, 0(s0)                # area = t6
    addi a7, zero, 10
    ecall
```

La asignación de puntaje se distribuirá de la siguiente forma:

- **1 pto.** por calcular correctamente la base. Se descuentan **0.5 ptos.** si no se obtiene su módulo.
- **1 pto.** por calcular correctamente la altura. Se descuentan **0.5 ptos.** si no se obtiene su módulo.
- **1 pto.** por calcular correctamente el área. Se descuentan **0.5 ptos.** si solo realiza la multiplicación y no la división por 2.