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

IIC2343 - Arquitectura de Computadores (II/2023)

Actividad práctica 2

Sección 1 - 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
    addi sp, sp, -4
    sw ra, 0(sp)
    call check_is_prime
    lw ra, 0(sp)
    addi sp, sp, 4
    sw t3, 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 t3, 1
      j end
    is_not_prime:
      li t3, 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 el valor de retorno no se encuentra en ninguno de los registros `a0`, `a1`. 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
    call pow
    sw a0, 0(s0)
    li a7, 10
    ecall
  pow:
    beq a1, s1, end
    addi sp, sp, -4
    sw a0, 0(sp)
    addi a1, a1, -1
    call pow
    lw t0, 0(sp)
    addi sp, sp, 4
    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 respalda la dirección de retorno en cada llamado recursivo de `pow`**. Basta con asegurar el respaldo del registro `ra` antes del llamado recursivo y su recuperación antes del retorno, 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 **ordene un arreglo `arr` de largo `len` y guarde su mediana en `median`**. El arreglo ordenado puede almacenarse en la variable original o en una variable nueva y, por otra parte, puede usar cualquier algoritmo de ordenamiento para llevar a cabo su implementación. Recuerde que la mediana de un arreglo corresponde a:

- El elemento central, si `len` $\% 2 = 1$.
- El promedio de los elementos centrales, si `len` $\% 2 = 0$.

Puede utilizar el siguiente fragmento de código como base:

```
.data
  arr:    .word 7, 3, 2, 1, 5
  len:    .word 5
  median: .word -1
.text
  # Su código aquí
```

La asignación de puntaje se distribuirá de la siguiente forma:

- **2 ptos.** por ordenar correctamente el arreglo. Se descuenta **1 pto.** si existe como máximo un error de implementación y no se asigna puntaje si existe más de uno.
- **1 pto.** por encontrar correctamente el elemento central según el largo del arreglo. 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 ordena el arreglo a partir de *bubble sort* y obtiene la mediana para el arreglo ordenado, ya sea con largo par o impar.

```
.data
  arr:    .word 7, 3, 2, 1, 5
  len:    .word 5
  median: .word -1
.text
  li s0, 2                    # Divisor para evaluar paridad y calcular media.
  la s1, arr                  # Dirección inicial del arreglo.
  li s2, 4                    # Constante para multiplicar índices por 4 para obtener direcciones.
  la s3, median               # Dirección de la mediana.
  lw s4, len
  mv a0, s1
  mv a1, s4
  addi sp, sp, -4
  sw ra, 0(sp)
  call sort_arr               # sort_arr(arr, len)
  lw ra, 0(sp)
  addi sp, sp, 4
  rem t0, s4, s0
  beqz t0, even_median
  odd_median:
    div t1, s4, s0
    mul t2, t1, s2
    add t2, t2, s1            # t2 = dir(arr) + 4 * median_index
    lw t2, 0(t2)              # t2 = arr[median_index]
    sw t2, 0(s3)              # median = arr[median_index]
    j end
  even_median:
    div t1, s4, s0
    mul t2, t1, s2
    sub t3, t2, s2
    add t2, t2, s1            # t2 = dir(arr) + 4 * right_central_index
    add t3, t3, s1            # t3 = dir(arr) + 4 * right_central_index - 4 = left_central_address
    lw t2, 0(t2)              # t2 = arr[right_central_index]
    lw t3, 0(t3)              # t3 = arr[left_central_index]
    add t4, t2, t3
    div t4, t4, s0            # t4 = (arr[left_central_index] + arr[right_central_index]) // 2
    sw t4, 0(s3)              # median = (arr[left_central_index] + arr[right_central_index]) // 2
  end:
    li a7, 10
    ecall
  sort_arr:
    li t0, 0                  # index i
    li t1, 0                  # index j
    addi t2, a1, -1           # t2 = len - 1
    i_loop:
      beq t0, t2, sort_end
      j_loop:
        mul t3, t1, s2        # t3 = 4*j
        add t3, t3, a0        # t3 = dir(arr) + 4 * j
        addi t4, t3, 4        # t4 = t3 + 4 = dirr(arr) + 4 * (j+1)
        lw t5, 0(t3)          # t5 = arr[j]
        lw t6, 0(t4)          # t6 = arr[j+1]
        bgt t5, t6, swap
        continue_j_loop:
          addi t1, t1, 1      # j += 1
          blt t1, t2, j_loop
          j continue_i_loop
        swap:
          sw t6, 0(t3)        # arr[j] = t6 = arr[j+1]
          sw t5, 0(t4)        # arr[j+1] = t5 = arr[j]
          j continue_j_loop
      continue_i_loop:
        li t1, 0              # Reseteo j
        addi t0, t0, 1        # i += 1
        j i_loop
    sort_end:
      ret
```

La asignación de puntaje se distribuirá de la siguiente forma:

- **2 ptos.** por ordenar el arreglo `arr`, independiente de si se realiza en la variable original o si se realiza en una nueva. Se descuenta **1 pto.** si se realiza algún tipo de ordenamiento, pero no de forma correcta. No se asigna puntaje si el arreglo no se modifica de ninguna forma.
- **1 pto.** por encontrar la mediana para arreglos de largo par o impar. Se descuentan **0.25 ptos.** si se obtienen los elementos centrales de un arreglo de largo par sin calcular de forma correcta su mediana; o bien si se obtiene la mediana pero no se almacena en la variable correspondiente. Se descuentan **0.5 ptos.** si solo se obtiene de forma correcta para arreglos de largo par o impar. No se otorga puntaje en cualquier otro caso.