color

IIC2343 – Arquitectura de Computadores

Pauta Control 2

Profesor: Yadran Eterovic

1. Subrutinas (3 puntos)

1. Siguiendo la ISA de la arquitectura MIPS, desarrolla los siguientes ejercicios en *assembly*:

   a. Defina la función $div(a, b)$, la cual recibe dos parámetros a y b, ambos potencias de 2 y retorna el cuociente a/b.

   Solución:

```
division:
    addi $v0, $zero, 0
loop:
    slt  $t0, $a0, $a1    # $t0 = $a0 < $a1 ? 1 : 0 (floor division)
  # slt  $t0, $a0, $zero  # $t0 = $a0 < 0 ? 1 : 0 (roof division)
    bne  $t0, $zero, end  # if $a0 < $a1, jump to end
    beq  $a0, $a1, end    # if $a0 == $a1, jump to end
    sub  $a0, $a0, $a1    # $a0 = $a0 - $a1
    addi $v0, $v0, 1      # $v0 = $v0 + 1
    j    loop
end:
    jr   $ra
```

   Asignacion de puntaje:

   - 1 punto por hacer una forma de división correcta.

   b. Escriba una función que, usando $div(a, b)$, calcule el cuociente entre 2 números enteros. (Puedes suponer que existe la función $div(a, b)$

   Solución:

```
top_division:
    addi $sp, $sp, -4  # assign stack space
    sw   $ra, 0($sp)   # save $ra value
    jal  division      # values $a0 and $a1 already loaded, $v0 gets loaded
    lw   $ra, 0($sp)   # load $ra value
    jr   $ra
```

   Asignacion de puntaje:

   - 0.5 puntos por correcto manejo del *stack*.
   - 0.5 puntos por correcto llamado a subrutina.
   - 0.5 puntos por guardar y volver a cargar el valor de r_a.

c. Usa tu respuesta anterior para dividir 32 en 4.

Solución:

```
addi $a0, $zero, 32    # load $a0 = 32
addi $a1, $zero, 4     # load $A1 = 4
jal  top_division      # call div(32, 4)
add  $t0, $zero, $v0   # not necessary
```

Asignacion de puntaje:

- 0.25 puntos por cargar los valores pedidos.
- 0.25 puntos por el llamado a subrutina.

2. Cree una función para calcular el factorial de un número recursivamente utilizando la ISA de la arquitectura MIPS. Considere que NO cuentas con una instrucción que permita multiplicar números y tampoco puedes usar *shifts*. (*Hint: sería más fácil tener una función que multiplicara, ¿verdad?*)

Solución:

```
factorial:
    beq  $a0, $zero, final  # si a0 == 0, llamamos a final
    addi $sp, $sp, -8       # guardamos espacio en el stack
    sw   $ra, 4($sp)        # almacenamos el valor de ra
    sw   $a0, 0($sp)        # almacenamos el valor de a0
    addi $a0, $a0, -1       # le restamos 1 al parametro a0
    jal  factorial          # llamada recursiva con a0 = a0 - 1
    lw   $a0, 0($sp)        # recuperamos el valor de a0
    add  $a1, $zero, $v0    # a1 = v0
    jal  multiplicacion     # subrutina para multiplicar a0 * v0 (a1)
    lw   $ra, 4($sp)        # recuperamos el valor de ra
    addi $sp, $sp, 8        # devolvemos espacio en el stack
    jr   $ra                # salimos de la subrutina
final:
    addi $v0, $zero, 1      # v0 = 1
    jr   $ra                # salimos de la subrutina


multiplicacion:
    addi $sp, $sp, -4       # guardamos espacio en el stack
    sw   $s0, 0($sp)        # almacenamos el valor de s0
    addi $s0, $zero, 0      # s0 = 0 (contador)
    addi $v0, $zero, 0      # limpiamos el registro de retorno
loop_mul:
    beq  $s0, $a0, end_mul  # si contador == a0, saltar al final
    addi $s0, $s0, 1        # s0 += 1 (aumentar contador)
    add  $v0, $v0, $a1      # v0 += a1
    j    loop_mul           # saltar a 7
end_mul:
    lw   $s0, 0($sp)        # recuperamos el valor de s0
    addi $sp, $sp, 4        # devolvemos el espacio en el stack
    jr   $ra                # salimos de la subrutina
```

Asignacion de puntaje:

- 1,5 pts por definir correctamente la subrutina *mul* (si se implementa la multiplicación de manera correcta sin el uso de subrutinas, se otorga puntaje completo).
  - 0,5 pts. uso correcto de subrutinas.
  - 0,5 pts. por correcto manejo del *stack* y recursión.
  - 0,5 punto por retornar el resultado correcto.
- 1,5 pts por definir correctamente la subrutina *factorial*.
  - 0,5 pts. uso correcto de subrutinas.
  - 0,5 pts. por correcto manejo del *stack*.
  - 0,5 pts. por retornar el valor correcto.

2. Punto Flotante (3 puntos)

1. Demuestra que los números floante del estándar IEEE 754 **no** cumplen con el principio de asociatividad de la suma, *i.e.*, $x + (y + z) = (x + y) + z$

   Solución: Los números del tipo float tienen problemas cuando las sumas incluyen números extremadamente dispares, *i.e.*, modulos muy grandes y muy chicos. Esto se debe a que el proceso de suma implica igualar los exponentes, proceso que puede llevar a que se trunquen elementos del número de menor módulo. Teniendo en cuenta, si se seleccionan los valores $x = -1,5 \times 2^{53}$, $y = 1,5 \times 2^{53}$, $z = 1,0$, tenemos lo siguiente en el lado izquierdo de la igualdad:

   $$x + (y + z) = -1,5 \times 2^{53} + (1,5 \times 2^{53} + 1,0) = -1,5 \times 2^{53} + 1,5 \times 2^{53} = 0$$

   En este caso, al igualar los exponentes en la suma de la derecha, el valor 1,0 se pierde, ya que el significante no tiene la cantidad de cifras necesarias. si analizamos ahora el lado derecho de la igualdad tenemos:

   $$(x + y) + z = (-1,5 \times 2^{53} + 1,5 \times 2^{53}) + 1,0 = 0 + 1,0 = 1,0$$

   A diferencia del caso anterior, acá no se pierde información, ya que la primera opción se realiza con números con exponente similar.

   Asignacion de puntaje:

   - 1,5 pts. por identificar el problema
   - Basta con encontrar números que cumplan lo pedido o cualquier demostración correcta para el puntaje completo.

2. Describe un algoritmo en pseudocódigo (o Python) que dado cierto binario de punto flotante retorne el número en base 10 según el estándar IEEE 754 visto en clases. Considera el *bit* de signo junto a los *bits* de fracción y exponente.

   Solución: Según el estándar IEEE 754, los números de punto flotante de 32 *bits* están compuestos de la siguiente forma:

   $$(-1)^{signo} \times (1 + fraccion) \times 2^{exponente-127}$$

   Donde se tiene un *bit* para el signo, 8 para el exponente y 23 para la fracción.

   Por lo tanto, el algoritmo pedido puede ser el siguiente:

```python
def float_to_decimal(num):

    sign = num[0]
    exp = int(num[1:9], 2) - 127
    frac = int(num[9:33], 2) * 2**(-23)

    value = (1 + frac) * 2**exp

    if sign == "1":
        return - value
    else:
        return value
```

   Asignacion de puntaje:

   - 1 punto por identificar cada parte del número de punto flotante correctamente.
   - 0.5 puntos por conversión binaria-decimal correcta del exponente.
   - 0.5 puntos por conversión binaria-decimal correcta de la fracción.
   - 1 punto por retornar correctamente el valor del número decimal.