Taller de Arquitectura - Programación

Sistemas Digitales

Primer Cuatrimestre 2025

Ejercicios

Ejercicio 1

Los siguientes programas fueron escritos por 2 programadores sin comunicarse. Programador **A** escribió las funciones etiquetadas como **FUNCION**, mientras que Programador **B** escribió sus casos de test. Tanto los testeos como las funciones debían utilizar la convención de llamada estándar, ya que luego se agregarían al resto del código de la empresa donde ambos trabajan. Aunque ambos dicen haber cumplido con esto, al evaluar, todos los test fallan.
Para cada programa:

- Comentar los casos de test y explicar que se está evaluando.
- Comentar el código de la función, explicar su funcionamiento y darle un nombre descriptivo.
- Marcar el prólogo y epilogo de la función.
- Encontrar los errores causados por no seguir la convención (no hay errores lógicos) y decidir si es culpa del Programador **A** y/o Programador **B**. Justificar.
- Arreglar la función, los casos de test y comprobar el funcionamiento en el emulador Ripes.

a)

```
1  main:        li s1, 2024
2               mv a0, s1
3               jal ra, FUNCION
4               add a0, s1, a0
5               bnez a0, noFunciona
6  funciona:    li a1, 1
7               j fin
8  noFunciona:  li a1, 0
9  fin:         j fin
```

```
1  FUNCION:  addi sp, sp, -4
2            sw ra, (0)sp
3            not s1, a0
4            addi a0, s1, 1
5            lw ra, (0)sp
6            addi sp, sp, 4
7            ret
```

b)

```
1  main:        li a0, 4
2               li a1, 6
3               jal ra, FUNCION
4               li a2, 10
5               bne a0, a2, noFunciona
6  funciona:    li a1, 1
7               j fin
8  noFunciona:  li a1, 0
9  fin:         j fin
```

```
1  FUNCION:  addi sp, sp, -4
2            sw ra, (0)sp
3            add a3, a0, a1
4            lw ra, (0)sp
5            addi sp, sp, 4
6            ret
```

c)

```
1   main:        li a0, 1
2                li a1, 2
3                jal ra, FUNCION
4                li a3, 3
5                bne a0, a3, noFunciona  # (4*1 - 2/2) != 3
6                li a0, 3
7                jal ra, FUNCION
8                li a3, 11
9                bne a0, a3, noFunciona  # (4*3 - 2/2) != 11
10               li a1, 12
11               jal ra, FUNCION
12               li a3, 6
13               bne a0, a3, noFunciona  # (4*3 - 12/2) != 6
14  funciona:    li a1, 1
15               j fin
16  noFunciona:  li a1, 0
17  fin:         j fin
```

```
1  FUNCION:  addi sp, sp, -4
2            sw ra, (0)sp
3            slli a2, a0, 2
4            srai a1, a1, 1
5            sub a0, a2, a1
6            lw ra, (0)sp
7            addi sp, sp, 4
8            ret
```

Ejercicio 2

Programe en lenguaje ensamblador de RISC-V las siguientes funciones y al menos 2 casos de test que compruebe el funcionamiento de cada una de ellas. Se debe usar la convención de llamada de RISC-V.

a) **Fibonacci Iterativo**

b) **Mayor en R^2**:

$$mayor(x_1, y_1, x_2, y_2) \begin{cases} 1 & \text{si } x_1 > x_2 \land y_1 > y_2 \\ -1 & \text{si } x_2 > x_1 \land y_2 > y_1 \\ 0 & \text{si no} \end{cases} \tag{1}$$

- Fin de checkpoint 1 -

Ejercicio 3

Los siguientes programas fueron escritos por 3 programadores sin comunicarse. Programador **A** escribió las funciones etiquetadas como **FUNCION** utilizando funciones auxiliares etiquetadas como **FUNCION_AUX**, provenientes de la biblioteca del Programador liechtensteiniano **B**, mientras que Programador **C** escribió sus casos de test. Tanto los testeos como las funciones debían utilizar la convención de llamada estándar y, según la documentación de la biblioteca, también lo debían hacer las funciones auxiliares. Aunque Programador **A** y **C** dicen haber cumplido con esto, al evaluar, todos los test fallan.
Para cada programa:

- Comentar los casos de test y explicar que se está evaluando.
- Comentar el código de la función y función auxiliar, explicar su funcionamiento y darle un nombre descriptivo.
- Marcar el prólogo y epilogo de cada función.
- Encontrar los errores causados por no seguir la convención (no hay errores lógicos, solo de convención o stack) y decidir cuáles programadores son los culpables. Justificar
- Arreglar las funciones, los casos de test y comprobar el funcionamiento en el emulador Ripes.
- Realizar un seguimiento del stack.

a)

```
1   main:        li a0, 4
2                li a1, 87
3                li a2, -124
4                li a3, -14
5                jal ra, FUNCION
6                li a2, -124
7                bne a0, a2, noFunciona
8   funciona:    li a1, 1
9                j fin
10  noFunciona:  li a1, 0
11  fin:         j fin
```

```
1  FUNCION:  addi sp, sp, -12
2            sw a2, (0)sp
3            sw a3, (4)sp
4            sw ra, (8)sp
5            jal ra, FUNCION_AUX
6            mv s1, a0
7            lw a0, (0)sp
8            lw a1, (4)sp
9            jal ra, FUNCION_AUX
10           mv a1, s1
11           jal ra, FUNCION_AUX
12           lw ra, (8)sp
13           addi sp, sp, 12
14           ret
```

```
1  FUNCION_AUX:  addi sp, sp, -4
2                sw ra, (0)sp
3                bgt a1, a0, terminar
4                mv a0, a1
5  terminar:     ret
```

b)

```
1  main:         li a0, 3
2                li a1, 10
3                li a2, -5
4                li a3, 2
5                li a4, 5
6                li a5, -1
7                jal ra, FUNCION
8                li a2, 1
9                bne a0, a2, noFunciona
10 funciona:     li a1, 1
11               j fin
12 noFunciona:   li a1, 0
13 fin:          j fin
```

```
1  FUNCION:  addi sp, sp, -12
2            sw a2, (0)sp
3            sw s0, (4)sp
4            sw ra, (8)sp
5            li s0, 1
6            mv a2, a4
7            jal ra, FUNCION_AUX
8            bne a0, s0, return
9            lw a0, (0)sp
10           mv a1, a3
11           mv a2, a5
12           jal ra, FUNCION_AUX
13           bne a0, s0, return
14           lw s0, (4)sp
15           lw ra, (8)sp
16           addi sp, sp, 12
17 return:   ret
```

```
1  FUNCION_AUX:  addi sp, sp, -4
2                sw ra, (0)sp
3                sub a3, a2, a0
4                blt a3, zero, afuera
5                sub a5, a2, a1
6                bgt a5, zero, afuera
7  adentro:      li a0, 1
8                j terminar
9  afuera:       li a0, 0
10 terminar:     lw ra, (0)sp
11               addi sp, sp, 4
12               ret
```

- Fin de checkpoint 2 -

Ejercicio 4

Programe en lenguaje ensamblador de RISC-V las siguientes funciones y al menos 2 casos de test que compruebe el funcionamiento de cada una de ellas. Se debe usar la convención de llamada de RISC-V.

a)
- **Inv(x)** = -x
- **InvertirArreglo**: Dado un puntero a un arreglo de enteros de 32 bits y la cantidad de elementos, cambia cada valor del arreglo por su inverso aditivo.

b)
- **EsPotenciaDeDos**

$$EsPotenciaDeDos(x) = \begin{cases} 1 & \text{si } \exists k \in N : 2^k = x \\ 0 & \text{si no} \end{cases} \tag{2}$$

- **PotenciasEnArreglo**: Dado un puntero a un arreglo de enteros sin signo de 8 bits y la cantidad de elementos, devuelve cuantos de ellos son potencias de 2.
ayuda: Pensar como una potencia de dos se ve en base binaria.

Ejercicio 5

Programe en lenguaje ensamblador de RISC-V las siguientes funciones recursivas y al menos 2 casos de test que compruebe el funcionamiento de cada una de ellas. Se debe usar la convención de llamada de RISC-V.

a) **Factorial**:

$$fact(x) = \begin{cases} 1 & \text{si } x = 0 \\ x \cdot fact(x-1) & \text{si no} \end{cases} \tag{3}$$

b) **Fibonacci_3**:

$$F_3(x) = \begin{cases} 0 & \text{si } x = 0 \\ 1 & \text{si } x = 1 \\ 2 & \text{si } x = 2 \\ F_3(x-1) + F_3(x-2) + F_3(x-3) & \text{si no} \end{cases} \tag{4}$$

- Fin de checkpoint 3 -

Ejercicio 6 Se tiene la estructura **InformacionAlumno** que contiene el ID del alumno y su nota en el ultimo examen, numeros sin signo de 16 bits y 8 bits respectivamente. En memoria se encuentra un arreglo del tipo InformacionAlumno con la forma:

Direccion	0x0000	0x0002	0x0003	0x0005	...	0x0030	0x0032	0x0033
Valor	5492	1	8886	6	...	6540	10	0

Donde el final del arreglo es demarcado por un ID nulo. Se pide

- Calcular cuantos bytes ocupa en memoria la estructura InformacionAlumno
- Escribir una funcion que dado un puntero a un arreglo de InformacionAlumno, devuelva la suma de las notas de los alumnos con ID impar. Escribir un caso de test donde verificar el funcionamiento de la funcion.

  Ayuda: Para crear el arreglo en Ripes pueden hacerlo definiendo por separado cada elemento de InformacionAlumno en .data
  Ejemplo:

```
.data
tablaCalificaciones:    .half 5523
                        .byte 3
                        .half 8754
                        .byte 6
                        ⋮
                        .half 0         #Declaramos el final del arreglo
```

- Fin de checkpoint 4 -