Práctica Programacion RISC-V

Sistemas Digitales

Primer Cuatrimestre 2025

Convencion de llamada

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
```

d)

```
1  main:         li a0, 4
2                li a1, 87
3                jal ra, FUNCION
4                li a2, 87
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              mv a0, a2
4              bgt a0, a5, terminar
5              mv a0, a5
6  terminar:   lw ra, (0)sp
7              addi sp, sp, 4
8              ret
```

e)

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

```
1  FUNCION:       addi sp, sp, -4
2                 sw ra, (0)sp
3                 mv a1, a0
4                 mv a0, zero
5  inicioCiclo:   beq a1, zero, finCiclo
6                 add a0, a0, a1
7                 addi a1, a1, -1
8                 j inicioCiclo
9  finCiclo:      lw ra, (0)sp
10                addi sp, sp, 4
11                ret
12
13
```

f)

```
1   main:           li a0,7
2                   li a1,13
3                   jal ra, FUNCION
4                   mv s1, a0
5                   li a1,-1
6                   jal ra, FUNCION
7                   beq a0, a2, equivalentes
8   diferentes:     li a1,0
9                   j fin
10  equivalentes:   li a1,1
11  fin:            j fin
```

```
1   FUNCION:        addi sp, sp, -4
2                   sw ra, (0)sp
3                   mv a2, a1
4                   bgt a1, zero, inicioCiclo
5                   sub a2, zero, a1
6   inicioCiclo:    blt a2, a0, finCiclo
7                   sub a2, a2, a0
8                   j inicioCiclo
9   finCiclo:       mv s1, a0
10                  mv a0, a2
11                  bgt a1, zero, terminar
12                  beq a0, zero, terminar
13                  sub a0, s1, a0              # (-n)%m = m - (n%m)
14  terminar:       lw ra, (0)sp
15                  addi sp, sp, 4
16                  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) **Multiplicacion**: $mult(x, y) = x \cdot y \ (x, y \in Z)$

b) **Fibonacci Iterativo**

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

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

d) **Division**: $div(x, y) = \lfloor \frac{x}{y} \rfloor \ (x, y \in Z)$

Uso del Stack

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
```

Ejercicio 4 Indique cuáles son las obligaciones y las garantías que tiene la funcion llamada y la llamadora. Explique en sus palabras porque cree que se usan estar reglas ¿Cuál es la utilidad de una convención de llamada?

Ejercicio 5

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.

c)
- **EvaluarMonomio**(x,c,p) = $c \cdot x^p$
- **EvaluarPolinomio**: Dado un puntero a un arreglo de enteros de 32 bits, la cantidad de elementos del arreglo y un entero x, evalua en x el polinomio construido usando como coeficientes los valores dentro del arreglo. Ejemplo:
  El arreglo

3	-1	5	0	2

  Equivaldria al polinomio

$$P(x) = 3 - x + 5 \cdot x^2 + 2 \cdot x^4$$

Recursion

Ejercicio 6

Los siguientes programas fueron escritos por 2 programadores sin comunicarse. Programador A escribió las funciones recursivas etiquetadas como FUNCION, mientras que Programador B escribió sus casos de test. 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 A dice 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.
- Indicar el Caso Base y la definicion recursiva de la funcion.
- Encontrar los errores causados por no seguir la convención o hacer mal uso del stack. Justificar.
- Arreglar la función y comprobar el funcionamiento en el emulador Ripes.
- Para un caso de test, realizar un grafico del flujo que realiza el programa.

a)

```
1  main:        li a0, 13
2               li a1, 5
3               jal FUNCION
4               li a1, 3                # 13 mod(5) = 3
5               bne a0, a1, noFunciona
6  funciona:    li a1, 1
7               j fin
8  noFunciona:  li a1, 0
9  fin:         j fin
```

```
1  FUNCION:   blt a0, a1, terminar
2             sub a2, a0, a1
3             jal FUNCION
4  terminar:  ret
```

b)

```
1  main:        li a0, 4
2               jal FUNCION
3               li a1, 5
4               bne a0, a1, noFunciona
5               li a0, 5
6               jal FUNCION
7               li a1, 8
8               bne a0, a1, noFunciona
9               li a0, 6
10              jal FUNCION
11              li a1, 13
12              bne a0, a1, noFunciona
13 funciona:    li a1, 1
14              j fin
15 noFunciona:  li a1, 0
16 fin:         j fin
```

```
1  FUNCION:    addi sp, sp, -8
2              sw a0, (0)sp
3              sw ra, (4)sp
4              li a1, 1
5              beq a0, zero, casoBase0
6              beq a0, a1, casoBase1
7              addi a0, a0, -1
8              jal FUNCION
9              mv a1, a0
10             lw a0, (0)sp
11             addi a0, a0, -2
12             jal FUNCION
13             add a0, a1, a0
14             j prologo
15 casoBase0:  li a0, 1
16             j prologo
17 casoBase1:  li a0, 1
18 prologo:    lw ra, (4)sp
19             addi sp, sp, 8
20             ret
```

c)

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

```
1   FUNCION:   beq a0, zero, casoBase
2              addi sp, sp, -8
3              sw a0, (0)sp
4              sw ra, (4)sp
5              addi a0, a0, -1
6              jal FUNCION
7              lw a1, (0)sp
8              add a0, a1, a0
9              j prologo
10  casoBase:  li a0, 0
11  prologo:   lw ra, (4)sp
12             addi sp, sp, 8
13             ret
```

Ejercicio 7 Indique cuantos bytes de espacio en el stack se utilizan para encontrar el quinto elemento de Fibonacci en su implementacion recursiva ¿Cuantos se utilizarian para encontrar el n-esimo elemento? ¿Cuantos bytes utiliza su implementacion iterativa?

Ejercicio 8

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) **Profundidad de Collatz**:
Segun la conjetura de Collatz, si se aplica la funcion

$$f(n) = \begin{cases} \frac{n}{2} & \text{si n es par} \\ 3n+1 & \text{si n es impar} \end{cases} \tag{4}$$

a un numero natural cualquiera, con suficientes repeticiones se llegara al numero 1.
Por ejemplo $6 \to 3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$.

- Implementar la funcion EsPar, que dado un numero devuelve 0 si es impar y 1 si es par.
- Utilizar EsPar para construir una funcion que dado un numero, devuelve la cantidad de repeticiones necesarias de f para llegar a 1.
  Por ejemplo: $Pc(1) = 0$ y $Pc(6) = 8$

c) **Fibonacci_3**:

$$F_3(x) = \begin{cases} 0 & \text{si } x = 0 \\ 1 & \text{si } x = 1 \\ 2 & \text{si } x = 2 \\ F_3(x-1) + F_3(x-2) + F_3(x-3) & \text{si no} \end{cases} \tag{5}$$

d) **Fibonacci_n**: n sera un argumento con la funcion asociada

$$F_n(x) = \begin{cases} x & \text{si } x < n \\ \sum_{i=1}^{n} F_n(x-i) & \text{si no} \end{cases} \tag{6}$$

e) **Raiz de una Funcion Lineal**:
Dado 3 argumentos: min, max y un puntero a una funcion lineal f. Escribir una funcion que devuelva 0 si f no contiene una raiz en el intervalo $[min, max]$ o la raiz en caso contrario. (*ayuda*: Utilizar la instruccion jalr para llamar a una funcion por puntero y el Método de bisección para encontrar la raiz)

Manejo De Estructuras

Ejercicio 9 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

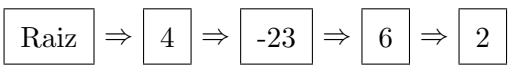
Donde el final del arreglo es demarcado por un ID nulo. Se pide

- 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
```

Ejercicio 10 Se tiene en memoria una **lista enlazada** de nodos, donde cada nodo contiene un valor de 32 bits en complemento a 2 y un puntero al siguiente elemento de la lista, el ultimo nodo contendra un puntero nulo (valdra 0) hacia el siguiente elemento. Por ultimo, se tendra una raiz, que apuntara al primer nodo de la lista, si su puntero fuera nulo, la lista estaria vacia.



Se pide:

- Calcular cuantos bytes ocupa en memoria un nodo ¿Cuanto ocupara una lista enlazada de n elementos?

- Escribir una funcion que dado un puntero a la raiz de una lista enlazada, devuelva la suma de los valores de cada nodo.

  Ayuda: Para crear una lista enlazada en ripes defina los nodos en el orden inverso de la lista, para asi poder hacer referncia a la etiqueta.
  Ejemplo:

```
.data
nodo_4:   .word -1 0x0
nodo_3:   .word 6 nodo_4
nodo_2:   .word 3 nodo_3
nodo_1:   .word -2 nodo_2
raiz:     .word nodo_1
```

Ejercicio 11 Se tiene en memoria una estructura **ArregloOrdenado**, compuesta por un puntero a un arreglo de numeros sin singo de 16 bits ordenado y la dimension de este arreglo representada en un numero de 32 bits en complemento a 2, donde el puntero sera nulo si la dimension es 0.

ArregloOrdenado	
0x10000018	4

Direccion	0x10000018	0x1000001A	0x1000001C	0x1000001E
Valor	2	54	1000	2500

Se pide:

- Calcular cuantos bytes ocupa en memoria la estructura ArregloOrdenado en su conjunto, incluyendo al arreglo asociado.
- Escribir una funcion que dado un puntero a un ArregloOrdenado y un valor, realice una busqueda binaria para encontrar y devolver el index del valor dentro del arreglo, o -1 si este no se encuentra. Implementar la busqueda de forma recursiva e iterativa.