Taller de Arquitectura - Programación

Sistemas Digitales

Segundo Cuatrimestre 2024

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. correciones

a) Se esta evaluando si la suma de un numero y su inverso aditivo es igual a 0.

```
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
```

s1 = 2024
a0 = s1 a0 = 2024
ra = pc
a0 = s1 + a0
si a0 != 0 vamos a noFunciona
a1 = 1 si funciona
a1 = 0 si no funciona
fin

Recibe un numero en a0 , lo invierte con un not y le suma uno obteniendo el inverso aditivo.

Inverso aditivo

```
1  FUNCION:  addi sp, sp, -4
2            sw ra, (0)sp
3  not a2, a0  not s1, a0
4  addi a0, a2, 1  addi a0, s1, 1
5            lw ra, (0)sp
6            addi sp, sp, 4
7            ret
```

sp = sp - 4
mem[sp] = ra
s1 = not a0
a0 = s1 + 1
ra = mem[sp]
sp = sp + 4
volvemos adonde este ra

La culpa es del programador A porque el error fue no volver s1 a su valor original ya que es uno de los registros que no se tienen que modificar. Si s1 hubiera quedado con su valor original el test hubiera funcionado.

La culpa es del programador A porque el error fue no devolver el resultado de la funcion en a0. Si hubiera sido asi se podria confirmar que la suma da igual al valor de a2.

b) Se esta evaluando si la suma de a0 y a1 es igual a a2, en este caso de test se le da a a2 el valor 10.

```
1  main:        li a0, 4                    a0 = 4
2               li a1, 6                    a1 = 6
3               jal ra, FUNCION             ra = pc
4               li a2, 10                   a2 = 10
5               bne a0, a2, noFunciona      si a0 != a2 vamos a noFunciona
6  funciona:    li a1, 1                    a1 = 1 si funciona
7               j fin
8  noFunciona:  li a1, 0                    a1 = 0 si no funciona
9  fin:         j fin                       fin
```

Hace la suma de a0 y a1.

```
   suma a0 y a1
1  FUNCION:  addi sp, sp, -4          sp = sp - 4
2            sw ra, (0)sp             mem[sp] = ra
3  add a0, a0, a1   add a3, a0, a1    a3 = a0 + a1
4            lw ra, (0)sp             ra = mem[sp]
5            addi sp, sp, 4           sp = sp + 4
6            ret                      volvemos adonde este ra
```

c) Se esta evaluando si la resta de los desplazamientos es igual a a3, en el primer caso a a3 se le da el valor 3, en el segundo 11 y en el tercero 6.

```
1  main:          li a0, 1                    a0 = 1
2                 li a1, 2                    a1 = 2
3                 jal ra, FUNCION             ra = pc
4  se agregan     li a3, 3                    a3 = 3
5  ambas instrucciones
                  bne a0, a3, noFunciona      # (4*1 - 2/2) != 3
6                 li a0, 3                    a0 = 3
7        li a1, 2 jal ra, FUNCION             ra = pc
8                 li a3, 11                   a3 = 11
9                 bne a0, a3, noFunciona      # (4*3 - 2/2) != 11
10       li a0, 3 li a1, 12                   a1 = 12
11                jal ra, FUNCION             ra = pc
12                li a3, 6                    a3 = 6
13                bne a0, a3, noFunciona      # (4*3 - 12/2) != 6
14 funciona:      li a1, 1                    a1 = 1 si funciona
15                j fin
16 noFunciona:    li a1, 0                    a1 = 0 si no funciona
17 fin:           j fin                       fin
```

Multiplica a0 por 4 despazandolo dos posiciones a la izquierda despues divide a1 por 2 desplazandolo una posicion a la derecha y por ultimo resta los resultados.

```
   Resta de desplazamientos
1  FUNCION:  addi sp, sp, -4          sp = sp - 4
2            sw ra, (0)sp             mem[sp] = ra
3            slli a2, a0, 2           a2 = a0*4
4            srai a1, a1, 1           a1 = a1/2
5            sub a0, a2, a1           a0 = a2 - a1
6            lw ra, (0)sp             ra = mem[sp]
7            addi sp, sp, 4           sp = sp + 4
8            ret                      volvemos adonde esta ra
```

La culpa es del programador B porque el error fue no guardar los valores de los registros temporarios a1 y a0 antes de la llamada.

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**:

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

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) Se esta evaluando si -124 es el minimo entre -124, -14, 4, 87.

```
1   main:        li a0, 4                  a0 = 4
2                li a1, 87                 a1 = 87
3                li a2, -124               a2 = -124
4                li a3, -14                a3 = -14
5                jal ra, FUNCION           ra = pc
6                li a2, -124               a2 = -124
7                bne a0, a2, noFunciona    si a0 != de a2 salta a noFunciona
8   funciona:    li a1, 1                  a1 = 1 si el programa funciona
9                j fin                     fin
10  noFunciona:  li a1, 0                  a1 = 0 si el programa no funciona
11  fin:         j fin                     fin
```

Primero compara los primeros dos parametros de entrada y guarda el valor del menor de ellos luego hace lo mismo con los siguientes dos parametros y por ultimo compara ambos resultados quedandose con el menor.

Si bien no afecta a este test, notamos que en esta funcion se modifica el valor de s1 y no se guarda su valor inicial.

guardamos en a4 el valor de s1 antes de que sea modificado por a0 y al final devolvemos el valor inicial de s1 guardado en a4.

Menor entre 4 numeros

```
1   FUNCION:   addi sp, sp, -12      sp = sp - 12
2              sw a2, (0)sp          mem[sp] =  a2
3              sw a3, (4)sp          mem[sp+4] = a3
4              sw ra, (8)sp          mem[sp+8] = ra
5   mv a4, s1  jal ra, FUNCION_AUX   ra = pc
6              mv s1, a0             s1 = a0
7              lw a0, (0)sp          a0 = mem[sp]
8              lw a1, (4)sp          a1 = mem[sp + 4]
9              jal ra, FUNCION_AUX   ra = pc
10             mv a1, s1             a1 = s1
11             jal ra, FUNCION_AUX   ra = pc
12             lw ra, (8)sp          ra = mem[sp + 8]
13  mv s1, a4  addi sp, sp, 12       sp = sp + 12
14             ret                   volvemos adonde esta el ra
```

Compara ambos parametros y se queda con el menor.

En esta funcion el error fue que falta el epilogo, quedaria terminar:
lw ra, 0(sp)
addi sp, sp, 4
ret

Menor

```
1   FUNCION_AUX:  addi sp, sp, -4      sp = sp - 4
2                 sw ra, (0)sp         mem[sp] = ra
3                 bgt a1, a0, terminar si a1 > a0 va a terminar
4                 mv a0, a1            a0 = a1
5   terminar:     ret                  volvemos adonde esta el ra
```

b) Se esta evaluando si 5 pertenece al intervalo [3,10] y si -1 pertenece al intervalo [-5,2].

```
1   main:         li a0, 3             a0 = 3
2                 li a1, 10            a1 = 10
3                 li a2, -5            a2 = -5
4                 li a3, 2             a3 = 2
5                 li a4, 5             a4 = 5
6                 li a5, -1            a5 = -1
7                 jal ra, FUNCION      ra = pc
8                 li a2, 1             a2 = 1
9                 bne a0, a2, noFunciona si a0 != a2 salta a noFunciona
10  funciona:     li a1, 1             a1 = 1 si funciona
11                j fin
12  noFunciona:   li a1, 0             a1 = 0 si no funciona
13  fin:          j fin                fin
```

El culpable del error es el programador A porque antes de llamar debe guardar los valores de los registros temporarios que necesite utilizar al retornar, en este caso serian a3 y a5 los que no guardo y despues utiliza en la segunda llamada. Ademas le falto poner la etiqueta de return antes del epilogo ya que sino todos los casos que cumplan la condicion se saltean el mismo.

Guarda en s0 un 1 que despues usa para chequear si la respuesta de la funcion auxiliar fue positiva o negativa, guarda y mueve los valores de los a para poder llamar a la funcion auxiliar y fijarse si pertenecen a los intervalos deseados.

ChequeoDeIntervalos

```
1   FUNCION:  addi sp, sp, -12      sp = sp - 12
2             sw a2, (0)sp          mem[sp] = a2
3   sw a3 (12)sp
    sw a5 (16)sp
              sw s0, (4)sp          mem[sp+4] = s0
4             sw ra, (8)sp          mem[sp+8] = ra
5             li s0, 1              s0 = 1
6             mv a2, a4             a2 = a4
7             jal ra, FUNCION_AUX   ra = pc
8             bne a0, s0, return    si a0 != s0 salta a return
9             lw a0, (0)sp          a0 = mem[sp]
10  lw a1 (12)sp  mv a1, a3         a1 = a3
    lw a2 (16)sp
11            mv a2, a5             a2 = a5
12            jal ra, FUNCION_AUX   ra = pc
13  return:   bne a0, s0, return    si a0 != s0 salta a return
14            lw s0, (4)sp          s0 = mem[sp+4]
15            lw ra, (8)sp          ra = mem[sp+8]
16            addi sp, sp, 12       sp = sp + 12
17  return:   ret                   vuelve adonde esta el ra
```

PertenecealIntervalo

```
1   FUNCION_AUX:  addi sp, sp, -4       sp = sp - 4
2                 sw ra, (0)sp          mem[sp] = ra
3                 sub a3, a2, a0        a3 = a2 - a0
4                 blt a3, zero, afuera  si a3 < 0 salta a afuera
5                 sub a5, a2, a1        a5 = a2 - a1
6                 bgt a5, zero, afuera  si a5 > 0 salta a afuera
7   adentro:      li a0, 1              a0 = 1
8                 j terminar            salta a terminar
9   afuera:       li a0, 0              a0 = 0
10  terminar:     lw ra, (0)sp          ra = mem[sp]
11                addi sp, sp, 4        sp = sp + 4
12                ret                   vuelve adonde esta el ra
```

Se fija si a2 - el valor mas chico del intervalo da positivo ya que si es asi significa que a2 es mas grande y por lo tanto hay chances que pertenezca al intervalo, despues para confirmar esto hace lo mismo pero con el valor mas grande del intervalo ya que si da negativo significa que a2 es mas chico y por lo tanto pertenece al intervalo.

- Fin de checkpoint 1 -

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**

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

- **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.

Seguimiento del stack de Ejercicio 3

3 a)

0x7ffffec	20	20	0	0	0
0x7ffffe8	-14	242	255	255	255
0x7ffffe4	-124	132	255	255	255
0x7ffffe0	64	64	0	0	0

0x7ffffec	20	20	0	0	0
0x7ffffe8	-14	242	255	255	255
0x7ffffe4	-124	132	255	255	255
0x7ffffe0	80	80	0	0	0

0x7ffffec	20	20	0	0	0
0x7ffffe8	-14	242	255	255	255
0x7ffffe4	-124	132	255	255	255
0x7ffffe0	88	88	0	0	0

3 b)

0x7ffffec	-1	255	255	255	255
0x7ffffe8	2	2	0	0	0
0x7ffffe4	28	28	0	0	0
0x7ffffe0	0	0	0	0	0
0x7ffffdc	-5	251	255	255	255
0x7ffffd8	88	88	0	0	0

0x7ffffff0	0	0	0	0	0
0x7ffffec	-1	255	255	255	255
0x7ffffe8	2	2	0	0	0
0x7ffffe4	28	28	0	0	0
0x7ffffe0	0	0	0	0	0
0x7ffffdc	-5	251	255	255	255
0x7ffffd8	108	108	0	0	0

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 2 -

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 3 -

Ejercicio 2

2 a)

```
1  main:
2  li a0, 3
3  jal ra, FIBONACCI
4  li s1, 2
5  bne s1, a0, noFunciona
6  li a0, 5
7  jal ra, FIBONACCI
8  li s1, 5
9  bne s1, a0, noFunciona
10 funciona: li a1, 1
11 j fin
12 noFunciona: li a1, 0
13 fin: j fin
14
15 FIBONACCI:
16 addi sp, sp, -4
17 sw ra, (0)sp
18 li t0, 1
19 li t1, 0
20 li t2, 2
21 addi a0, a0, 1
22 for: bge t2, a0, final
23 add a1, t0, t1
24 mv t1, t0
25 mv t0, a1
26 addi t2, t2, 1
27 j for
28 final:
29 mv a0, t0
30 lw ra, (0)sp
31 addi sp, sp, 4
32 ret
```

Hacemos caso de test para n = 3 y n=5 donde el resultado deberia ser 2 y 5 respectivamente

Funcion fibonacci iterativa

t0 es el caso base de 1
t1 es el caso base de 0
t2 es el iterador

2 b)

```
main:
li a0, 4
li a1, 3
li a2, 2
li a3, 1
jal ra, MAYOR
li a4, 1
bne a0, a4, noFunciona
li a0, 1
li a1, 2
li a2, 3
li a3, 4
jal ra, MAYOR
li a4, -1
bne a0, a4, noFunciona
li a0, 4
li a1, 2
li a2, 5
li a3, 1
jal ra, MAYOR
li a4, 0
bne a0, a4, noFunciona
li a0, 4
li a1, 2
li a2, 4
li a3, 3
jal ra, MAYOR
li a4, 0
bne a0, a4, noFunciona
funciona: li a1, 1
j fin
noFunciona: li a1, 0
fin: j fin

MAYOR: addi sp, sp, -4
sw ra, (0)sp
beq a0, a2, cero
beq a1, a3, cero
blt a0, a2, posmenos
blt a3, a1, uno
j cero
posmenos: blt a1, a3, menosuno
cero: li a0, 0
j final
uno: li a0,1
j final
menosuno: li a0, -1
final:
lw ra, (0)sp
addi sp, sp, 4
ret
```

Probamos un caso de test para cada una de las posibles respuestas y una para cuando los parametros de entrada son iguales

Fuimos haciendo if para las distintas condiciones

4 a) En el ejercicio 4 a y b no nos funcionan los casos de test. Por una razon que no pudimos resolver hay algunas lineas del codigo que el programa no ejecuta aunque no haya ningun salto de por medio.
No consultamos por mail porque el problema necesitaba verse en persona.

```
main:
li a0, 3
li t6 , 2
mv s0 a0
li a1, 92
li s1, 3
li s4 104
sw s1 0(s4)
sw s1, 0(a1)
li a2, 96
li s2, 4
li s5 108
sw s2 0(s5)
sw s2, 0(a2)
li a3, 100
li s3, 5
li s6 112
sw s3 0(s6)
sw s3, 0(a3)
jal ra, INVERTIRARREGLO
li t0, 0
for:
bge t0, s0, final
slli t1, t0, 2
add t3 , t1 , s4
add t5, t1, a0
lw t2 0(t5)
lw t4 0(t3)
add t2 t2 t4
bnez t2 noFunciona
addi t0, t0, 1
j for
final:
funciona:
li a1, 1
j fin
noFunciona:
li a1, 0
fin: j fin
```

Creamos el arreglo dos veces , una para invertirlo y otra para luego testear la funcion y ver si todas las sumas dan cero

```

INVERTIRARREGLO:
addi sp, sp, -4
sw ra, (0)sp
li t0, 0
mv t3, a1
mv t4 a0
fer:
bge t0, t4, end
slli t1, t0, 2
add t1, t1, a1
lw a0 0(t1)
jal ra INVERSO
sw a0, 0(t1)
addi t0, t0, 1
j fer
end:
mv a0, t3
lw ra, (0)sp
addi sp, sp, 4
ret

INVERSO:
addi sp, sp, -4
sw ra, (0)sp
not a2, a0
addi a0, a2, 1
lw ra, (0)sp
addi sp, sp, 4
ret
```

Calcula el inverso aditivo de todas las posiciones de un arreglo

Inverso aditivo

4 b)

```
main:
li a0,3
mv s0,a0
li a1,92
li s1,3
sw s1,0(a1)
li a2,96
li s2,4
sw s2,0(a2)
li a3,100
li s3,5
sw s3,0(a3)
jal ra,POTENCIAENARREGLO
li t0,1
bne t0,a0,noFunciona
funciona:
li a1,1
j fina
noFunciona:
li a1,0
fina: j fina

POTENCIAENARREGLO:
addi sp,sp,-4
sw ra,(0)sp
li t1,0
mv t2,a0
mv t3,a1
li t5,0
for:
bge t1,t2,final
slli t4,t1,2
add t4,t4,t3
lw a0,0(t4)
jal ra,ESPOTENCIA
add t5,t5,a0
addi t1,t1,1
j for
final:
mv a0,t5
lw ra,(0)sp
addi sp,sp,4
ret
```

Calculamos cuantos numeros del arreglo son potencia de dos.

```

ESPOTENCIA:
addi sp,sp,-4
sw ra,(0)sp
li t0,1
while:
bge t0,a0,fin
slli t0,t0,1
j while
fin:
beq t0,a0,uno
li a0,0
j end
uno:
li a0,1
end:
lw ra,(0)sp
addi sp,sp,4
ret
```

Chequeamos si un numero es potencia de dos o no. Devolvemos 1 si la respuesta es verdadera y 0 si es falsa.

5 a)

```
main:
li a0, 4
jal ra, factorial
li a2, 24
bne a0, a2, noFunciona
li a0, 5
jal ra, factorial
li a2, 120
bne a0, a2, noFunciona
funciona: li a1, 1
j fin
noFunciona: li a1, 0
fin: j fin

factorial:
addi sp,sp, -16
sw a0, 4(sp)
sw ra, 0(sp)
bnez a0, else
addi a0,zero,1
addi sp,sp ,16
jr ra
else:
addi a0 a0 -1
jal factorial
lw t1 4(sp)
lw ra, 0(sp)
addi sp sp 16
mul a0 , t1 , a0
jr ra
```

Probamos caso de test con n=4 y n=5 , donde el resultado deberia ser 24 y 120 respectivamente

Calculamos el factorial recursivamente.

5 b)

```
main:
li a0,5
jal ra,Fibonacci
li a2,11
bne a0,a2,noFunciona
li a0,3
jal ra,Fibonacci
li a2,3
bne a0,a2,noFunciona
funciona:
li a1,1
j fin
noFunciona:
li a1,0
fin: j fin

Fibonacci:
addi sp,sp,-20
sw a0,4(sp)
sw ra,0(sp)
bnez a0,noescero
lw ra,0(sp)
addi sp,sp,20
jr ra

noescero:
addi t0,zero,1
bne a0,t0,noesuno
lw ra,0(sp)
addi sp,sp,20
jr ra

noesuno:
addi t0,zero,2
bne a0,t0,else
lw ra,0(sp)
addi sp,sp,20
jr ra

else:
addi a0,a0,-1
jal Fibonacci
sw a0,8(sp)

lw a0,4(sp)
addi a0,a0,-2
jal Fibonacci
sw a0,12(sp)

lw a0,4(sp)
addi a0,a0,-3
jal Fibonacci
sw a0,16(sp)

lw a1,8(sp)
lw a2,12(sp)
lw a3,16(sp)
add a0,a1,a2
add a0,a0,a3
lw ra,0(sp)
addi sp,sp,20
jr ra
```

Calculamos fibonacci recursivo, donde primero chequeamos los casos base y despues en el caso recursivo gusradamos cada resultado en una direccion de memoria para depues sumar los resultados.

6

```
.data
tablaCalificaciones:
    .half 5523
    .byte 3
    .half 8754
    .byte 6
    .half 6577
    .byte 4
    .half 0

.text
.globl main
main:
la a0, tablaCalificaciones
jal ra notasimpares
li a5 7
bne a5 a0 noFunciona
funciona:
li a1 1
j end
noFunciona:
li a1 0
end: j end
```

```

notasimpares:
addi sp, sp, -4
sw ra, 0(sp)
li a1 0
while:
mv a2 a0
lh a0 0(a0)
beq a0 x0 fin
jal ra esimpar
beq x0 a0 espar
lb a3 2(a2)
add a1 a1 a3
espar:
addi a0 a2 3
j while
fin:
mv a0 a1
lw ra, 0(sp)
addi sp, sp, 4
ret

esimpar:addi sp, sp, -4
sw ra, 0(sp)
andi a0, a0, 1
lw ra, 0(sp)
addi sp, sp, 4
ret
```

La estructura de informacion de alumnos va a ocupar 3 bytes por cada alumno , dos para el ID y uno para la nota. Por lo tanto la cantidad de bytes que ocupa en memoria esta estructura seria 3*cantidad de alumnos. Vamos iterando en el arreglo de a tres posiciones fijandonos si el ID es impar y si es asi sumamos la nota.