

1020/22

Nro. ord.	Apellido y nombre	L.U.	#hojas
97	Andrés, Sebastián Tyrocco	1020/22	4

SISTEMAS DIGITALES - Parcial
Primer Cuatrimestre 2024

Ej. 1	Ej. 2	Ej. 3	Ej. 4	Nota
B	B	B	B	A+

Correctorx:

M. DíENA

Aclaraciones

- Anote apellido, nombre, LU y numere *todas* las hojas entregadas, entregando los distintos ejercicios en hojas separadas.
- Cada ejercicio será calificado con una de las siguientes tres notas: Bien, Regular o Mal. La división de los ejercicios en incisos es meramente orientativa. Los ejercicios se calificarán globalmente.
- El parcial **no es a libro abierto** pero pueden utilizar la cartilla de referencia entregada por la materia.
- **Importante:** Justifique sus respuestas.
- Un resultado sin suficiente justificación equivale a un ejercicio no resuelto.
- El parcial se aprueba con al menos dos ejercicios Bien y uno Regular. Para obtener un Regular es necesario demostrar conocimientos sobre el tema del ejercicio. Para la promoción deben contar con al menos tres ejercicios bien y uno regular.

Ejercicio 1 Se cuenta con dos datos sin signo de dos bytes cada uno almacenados en el registro s0 y queremos invertir su posición, esto quiere decir poner los 16 bits más altos en la parte baja y los 16 bits más bajos en la parte alta. Escriba un programa de ensamblador RISC V que realice esta operación y almacene el resultado en el registro a0.

Ejemplo:

Bits	31	16	15	0
s0	[0x1A90]	0x0200		

Cada byte del registro debería tener la forma 0x0200(1A90).

Ejercicio 2 Implemente la función rec en el lenguaje ensamblador RISC V de forma recursiva, respete la convención de llamada presentada en la materia, explique el uso que le dará a cada registro y cómo se asegura que sus valores se preservan antes y después de cada llamada a función.

$$rec(n) = \begin{cases} 0, & \text{si } n = 1 \\ 2 * n + rec(n - 1), & \text{si } n > 1 \end{cases}$$

Guía de resolución (opcional):

- Escriba una versión de pseudocódigo.
- Transforme cada caso a su equivalente de operaciones atómicas (descomponga las operaciones lógicas, aritméticas y llamadas a función).
- Identifique los registros a emplear para cada dato.
- Si debe preservar algún registro para respetar la convención, indique qué mecanismo utilizará.
- Defina un flujo de ejecución tentativo.

Ejercicio 3 Un servidor de un juego multiusuario mantiene una lista de los puntajes más altos en un arreglo de enteros de 1 byte sin signo. Queremos agregar lógica para determinar la cantidad de puntajes que tengan un valor mayor a 0xF0, que es el máximo valor alcanzable en una partida con la intención de detectar puntajes espúrcos.

Se cuenta con un arreglo **puntajes** de datos de 8 bits sin signo empaquetados de forma contigua. El largo del arreglo (en bytes) se define en la constante **largo**.

Escriba un programa que detecte si algún puntaje se encuentra por sobre el valor 0xF0. Si algún puntaje cumple con esta condición debemos poner un 1 en el registro **a0**, en caso contrario debemos poner un 0.

Ejemplo:

Dirección puntajes	0x00000000 0x07	0x00000001 0xB0	0x00000002 0xF1	0x00000003 0x07
-----------------------	--------------------	--------------------	--------------------	--------------------

En este caso debemos poner un 1 en a0 porque el segundo dato vale más que 0xF0.

Esqueleto de programa:

```

1 .data:
2 puntajes: .byte 0x07 0xF1 0xB1 0x07
3 largo: .byte 4
4
5 .text:
6 # Escribir el programa aca.

```

Ejercicio 4 ¿Qué significa Position Independent Code (PIC)? ¿Cómo afecta esto a los saltos (tanto condicionales como incondicionales) que usan etiquetas? ¿Se ven afectados estos saltos si se cambia la posición de la cual comienza el programa? ¿Qué puede suceder si mientras escribimos nuestro código en las instrucciones tipo J y B usamos inmediatos en vez de etiquetas?

Orden 97

- ① Dos datos sin signo en \$0, fluíremos invertir su posición.

$$S_0 \leftarrow 0x\overbrace{02}^{\text{dato1}}\overbrace{00}^{\text{dato2}}1A90$$

$$S_0' \leftarrow 0x\overbrace{1A}^{\text{dato1}}\overbrace{90}^{\text{dato2}}0200$$

Para realizar esto, podemos usar una máscara y realizar desplazamientos.

La idea es ...

$$S_1 = 0x0000FFFF \quad \leftarrow \text{Máscara}$$

Si hacemos $(S_0 \wedge S_1)$ y guardamos el resultado en t_0 , obtendremos el dato 2.

$$(S_1 \wedge S_0) = 0x00001A90 = t_0$$

Luego, podemos desplazar S_0 a la derecha 2 bytes (esto es 16 bits) con

$$\text{srl}i\ S_0\ S_0\ 16 \quad \# \text{sin signo (cero rizp)}$$

y nos queda ...

$$S_0' = 0x\overbrace{00}^{\text{dato1}}\overbrace{00}^{\text{dato2}}200$$

No puedo desplazar t_0 a la izquierda 16 bits (s/ signo)

$$\text{slli}i\ t_0\ t_0\ 16$$

$$t_0' = 0x1A90 = 0x\underbrace{0000}_{\text{rizp}}\overbrace{1A90}^{\text{dato1}}0000$$

finalmente con un or o add obteneremos free.

$$S_0'' = S_0' \text{ or } t_0$$

En RISC V esto quedó ..

lui \$1 15

cargo los 15 bits significativos

addi \$1 \$1 -1

cargo los 12 bits restantes

And to \$0 \$1

to $\leftarrow 0x00001A90$

slli \$0 \$0 16

\$0 $\leftarrow 0x00000200$

slli to to 16

to $\leftarrow 0xA9000000$

ADD \$0 to \$0

\$0 $\leftarrow 0xA9000200$

(Asumiendo $S_0 = 0x02001A90$)

Obs:

Como S_1 se ve del rango de 12 bits inmediatos, lo cargo con 2 operaciones

lui \$1 15

$S_1 = 0x0000F \dots$

Addi \$1 \$1 -1

$0x0000F000 + 0x00000FFF =$

0x0000FFFF

(1) (2) (3)

~~0x0000000000000001
+ 1111111111111111

0x0000FFFF~~

en las inmediatas
(12 bits)

$-1 \approx 0xFFFF$

(b) P²

orden 97

Ej 2 Implementar recursivamente

$$\text{rec}(n) = \begin{cases} 0 & n=1 \\ 2n + \text{rec}(n-1) & n>1 \end{cases}$$

- Planteo la idea en código de alto nivel:

+ int rec(int n) {

 if ($n=1$) {
 return 0
 } return $2*n + \text{rec}(n-1)$ # cRecursivo

}

- Para facilitar el pasaje a RISC V lo reescribo así:

+ int rec(int n) { # ($n>1$) \leftrightarrow not ($n\leq 1$) if ($n>1$) { t = $2 * n$ h = rec($n-1$)

h = h + t

return h

}

return 0

Caso Recursivo

Caso Base

}

- Voy a usar los sig registros -

A₀ \rightsquigarrow n. (Argumento a la función)A₂ \rightsquigarrow resultado (APRt)T₀ \rightsquigarrow $2 * N$ j T₃ \rightsquigarrow N (temporal)T₁ \rightsquigarrow 1 Cte T₂ \rightsquigarrow 2 Cte

Código Recurso

- Text: $L_1 + t_1 \cdot 1$

$L_1 + t_2 \cdot 2$

$L_1 A_2^2 \cdot 0$

Jal ra foo ✓

j 0

foo :

ADDi sp sp -16 ✓

Sw A0 0(sp) ✓

sw ra 4(sp) ✓

BGT A0 t1 Recursion ✓

L A2 0

j endfoo

Recursion

Mv t3 A0

ADDi A0 A0 -1

Jal ra foo

Mul t0 t3 t2

ADD A2 A2 t0

j endfoo

endfoo:

lw ra 4(sp)

lw A0 0(sp)

ADDi sp sp 16

jr ra

$t_1 \leq 1$

$t_2 \leq 3$

$A_0 \sim N = 3$ (arbitrario)

"recu(N)" ~ foo()

loop infinito

Pido memoria para guardar A0 y el RA en el stack frame (convención)

valido el caso recursivo
 $N \geq 1 \iff \text{not}(1 > N)$

} CASO BASE ($A_2 \leq 0$)

$t_3 \leq N$

$N' = N - 1$

"foo(N-1)" ✓

$t_0 = 2 * n$

$A_2 = \underbrace{\text{recu}(n-1)}_{\text{Aca' } A_2 \text{ guarda } \text{recu}(N-1)} + 2n$

endfoo se encarga de recuperar la memoria utilizada en los llamados recursivos (el stack frame)

↑↑↑
 Necesario por convenciones de llamados.

Bien

[E3] Lista de puntos más altos en un arreglo de enteros. + 1 byte de signo.

- Puntos.
- Largo.

Sea $P = \#\{x \in \text{Puntos} \mid x > 0 \times F0\} > 0$

$$(P \rightarrow A_0 = 1) \wedge (\neg P \rightarrow A_0 = 0)$$

Idea en Pseudo código.

```
* Void hayError (Array<int> Puntos, int largo, int res)
{
    int i := 0; int umbral = 240; res = 0;
    while (i < largo) {
        if (Puntos[i] > umbral) {
            res = 1;
            Break;
        }
        i++;
    }
}
```

$$\#(i < \text{largo}) \leftrightarrow \neg(i \geq \text{largo})$$

para el BGT.

3
3
3
3
3
3

O| Vale que $0xF0$ —> 00000000000000000000000000000000 ... 11100000
R| ^{ext sin signo}
S| ^{100 (ceros)}

32 bits

$$\text{O sea } \text{umbral} = 2^4 + 2^5 + 2^6 + 2^7 = 240$$

Registers | :

$$\left. \begin{array}{l} t_0 = i, t_1 = \& \text{ARR}[i], t_2 = \text{umbral}, t_3 = \text{ARR}[i] \\ A_1 = 10100, A_0 = \text{res} \end{array} \right\}$$

Códigos Disc V: | # $t_0 = i$, $t_1 = \&Arr[1]$, $t_2 = \text{UMARL}$, $t_3 = Arr[i]$
| $A1 = \text{largo}$, $A0 = \text{res}$

• Text:

Li t_2 240

$t_2 \leftarrow \text{UMARL}$

LA t_1 Rn_{rajes}

$t_1 \leftarrow \&Arr[0]$ ✓

LB A_1 LARGO

$A_1 = |Arr| = \text{largo}$

WLI A_0 0

$\text{res} = 0$

Li t_0 0

$i := 0$

jr r₂ while

loop inf (NPF)

while: BGt $t_0 \leq A_1$ fin # $i \geq \text{largo} \rightarrow \text{fin}$

LB t_3 0(t_1) # $t_3 = Arr[i]$

BGt $t_3 \neq t_2$ error # if ($Arr[i] > \text{UMARL}$)
→ error

ADDI t_1 $t_1 + 1$

$i++ \sim \text{new arr} += 1$

ADDI t_0 $t_0 + 1$

"vuelta al while".

j while

error:

Li A_0 1

$\text{res} = 1$

j fin

"Break"

fin:

jr r₂

fin \ Break

(fin xp)
(fin byes).

• data:

Rn_{rajes} .byte 0x07 0xF1 0xB1 0x07

largo : .byte 4.

b

Orden 97

[E4] ¿Qué significa PIC? (...

- Position Independent Code significa que los desplazamientos en las instrucciones de saltos (`branch`) son relativos.

Eso es, en el ensamblado se "traducen" los etiquetas a los saltos de memoria (en bytes) necesitados para ir a la instrucción pedida, desde el PC actual.

o8 No se van afectados los saltos si se cambia la posición desde la cual se ejecuta el programa (sigue funcionando!)

Si en cambio, RISC-V no siguiera esta regla y los saltos fueran "directos" apuntaran a la dirección de memoria particular,

→ En distintas ejecuciones el programa podría fallar al hacer saltos (Instrucciones `J` y `B`):

B

~~Esto habría sido un problema de dirección de memoria fija~~