Nro. ord.	Apellido y nombre	L.U.	#hojas
[redacted]		[redacted]	4

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

Correctorx: EDGAR

Sistemas digitales - **Parcial**

Primer Cuatrimestre 2024

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

Con este dato el registro debería valer 0x02001A90.

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 espúreos.

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	0x00000000	0x00000001	0x00000002	0x00000003
puntajes	0x07	0xB0	0xF1	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?

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

```
# precondition:  n >= 1
lw a2  n  ✗       (1)
li a3  1  ✓       (2)
jal ra rec ✓      (3) guardo el próximo valor del PC en ra
(nop)                 y en su lugar pongo el valor de la
rec:                  etiqueta rec
   beq a2 a3 base     (4)
   addi sp sp -16 ✓   (5) reservo espacio en el stack
   sw a2 0(sp) ✓      (6)   modificando su puntero
   sw ra 4(sp) ✓      (7)
   addi a2 a2 -1 ✓    (8) sumo el valor de a2 un inmediato = -1
   jal ra rec ✓       (9) realizo la recursión para el siguiente
   lw t0 0(sp) ✓      (10)   valor de n
   lw ra 4(sp) ✓      (11)
   addi sp sp 16 ✓    (12) libero la memoria que había reservado
   slli t1 t0 1 ✓     (13)   en la pila con anterioridad
   add a0 a0 t1 ✓     (14) sumo el registro t1 con a0 y
   jr ra ✓  RET       (15) salto   lo guardo en a0
                          a la posición definida en ra
base:
   xor a0 a0 a0 ✓     (16) limpio el valor de a0 dejándolo
   jr ra ✓  RET       (17) = (15)   en 0
```

para cada n > 1

M VIENE EN A_0 POR CONVENCIÓN DE LLAMADA

(1) en a2 estoré n, que me servirá para decrementarlo, posibilitando la recursión y su correspondiente corte al llegar al caso base n = 1 → 0

(2) en a3 cargo mi n caso base para tenerlo de referencia y poder comparar los sucesivos valores de n contra él para realizar el corte de la recursividad

(4) comparo el valor del registro a2 con el valor del

1). data

w: .word 0x1A900200 # Inicio la constante w

EL DATO VIENE EN S0

.text

① ~~lw a2 w~~ ✓ ① # cargo la palabra que hay en la dirección w en el registro a2 porque es negativo

② slli t0 a2 16 ✓ ② # realizo un cambio lógico hacia la izquierda de 16 posiciones para quedarme con los bits 0-15 en 16-31

③ srli t1 a2 16 ✓ ③ # realizo un cambio lógico a derecha, de esta manera los bits 16-31 pasan a estar en las posiciones 0-15. Utilizo siempre shift lógicos para que trate los valores como unsigned y de este modo deje los bits a izquierda en 0 (a diferencia del shift aritmético que podría dejarlos en 1)

④ or a0 t0 t1 ✓ ④ Finalmente con la parte alta en la parte baja en t1 y la parte baja en la parte alta en t0 hago un or entre ellos asignando el resultado en a0

16 bits ← 1A900200 → 02001A90 ②

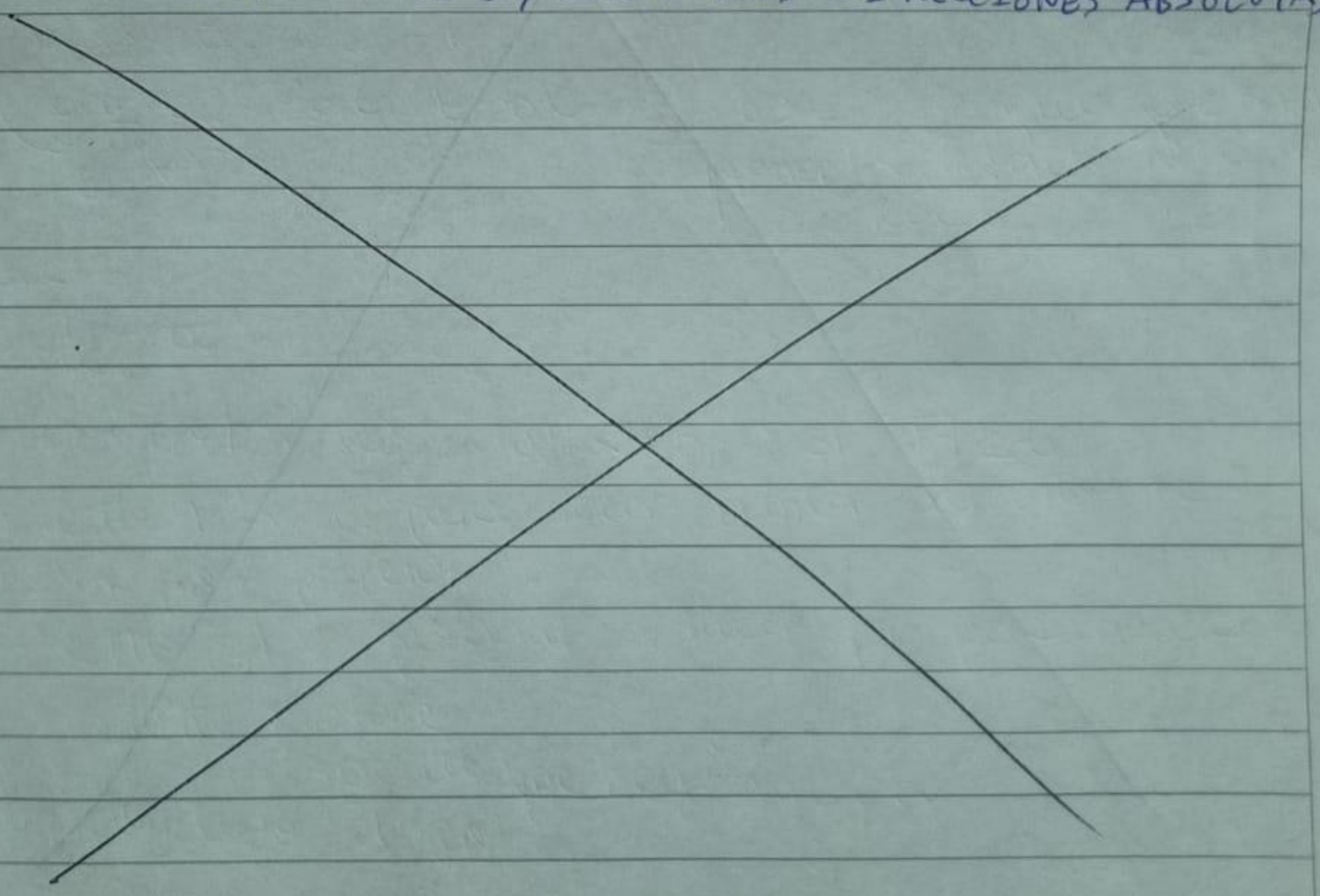
16 bits → 1A900200 → 00001A90 ③

or 00001A90 + 02000000 = 02001A90 ④

		4

4) Positun Independent Code (PIC) significa Código de Posición Independiente, esto es, código que sin importar su posición en memoria sabe resolver bien tanto sus saltos condicionales como incondicionales que utilizan etiquetas. En otras palabras, Los saltos hacia etiquetas pueden apuntar a posiciones fijas de memoria, por lo que estos saltos no se verán afectados si se cambia la posición en la cual comienza el programa. ✓
En el caso de utilizar inmediatos en vez de etiquetas al escribir código con instrucciones de tipo J y B lo que podría suceder es que al cambiar la posición de inicio del programa terminemos en una dirección que no es la queríamos (tanto dentro como incluso fuera del programa). ✓
nvólido

EL PROBLEMA NO SON LOS INMEDIATOS YA QUE SON OFFSETS, SINO LAS DIRECCIONES ABSOLUTAS



3) .data

```
    puntajes: .byte 0x07 0xF1 0xB1 0x07
    largo: .byte 4                          SON BYTES!
    referencia: .byte 0xF0
.text        ✓ #precondition: largo >=1 ?
    xor a0 a0 a0 ✓  ① limpio el registro a0
    la a2 puntajes  ② cargo la dirección de puntajes
 ✗ lb  ~~lw~~ a3 largo      ③ cargo el valor de largo
 ✗ lb  ~~lw~~ a4 referencia ④ cargo el valor de referencia
    jal ra for ✓    ⑤ guardo la próxima instrucción en ra
    (nop)              y me voy al for
for:
 ✗ lbu ~~lw~~ t0 0(a2)   ⑥ cargo en t0 el valor que hay en
    slt a0 a4 t0 ✓ ✓ ⑦     la dirección que tiene a2
    bne a0 zero break ⑧ si a0 ≠ 0 voy a break
    addi a2 a2 1 ✓     ⑨ sumo 1 a a2 para procesar
    addi a3 a3 -1 ✓ ✓  ⑩ el siguiente byte, es decir, puntaje
    beq a3 zero break  ⑪ si a3 = 0 significa que procesé
    j for ✓            ⑫ todos los puntajes, entonces voy
                          a break
break:
    ~~jr ra~~ ✓ ⑬
    RET
```

Si movía esa línea ahí es más eficiente ❊

⑦ asigno 1 en a0 si a4 < t0, 0 si a4 ≥ t0

⑩ resto 1 a a3 para saber cuántos elementos me quedan por procesar cual contador decreciente

⑫ salto al ~~x~~ próximo ciclo del for para procesar el siguiente puntaje

⑬ voy a la dirección que hay en ra, es decir, concluyo el for

❊ pero no quería llevar el parcial de liquidpaper

```
addi a3 a3 -1
beq a3 zero break
addi a2 a2 1
j for
```

registro a3 y de ser iguales voy al caso base
⑥ ∧ ⑦ guardo el valor del registro a2 y el valor de ra en la pila para utilizarlos más adelante
⑩ ∧ ⑪ recupero el último valor del último n° y el último ra (habiendo excluido la recursión luego de llegar al caso base) en ra
⑬ multiplico t0 (que tiene el valor del n actual) por 2, o lo que es lo mismo, realizo un shift lógico a izquierda de 1 posición (considerando que n>=1 por precondición) y lo guardo en t1

• a2 → {n, n-1, n-2, ..., 1} • a3 → 1 (caso base)

• t0 → recupero el anterior valor de a2
• t1 → guardo t0 * 2
• a0 → guardo la serie de los sucesivos valores de t; dios es el valor de retorno
• sp → se posibilita reservar memoria teniendo el puntero a la pila, para guardar los sucesivos valores de a2 (para saber qué valores procesar) y de ra (para saber a dónde volver)
• ra → guardo el valor de la siguiente instrucción del PC (que pisé con la mía)

Los valores se preservan antes y después de cada llamado a función porque se guardan en el stack y luego los obtengo nuevamente. Tanto el valor para preservar como el valor de la siguiente dirección de instrucción a ejecutarse