Nro. ord.	Apellido y nombre	L.U.	#hojas

SISTEMAS DIGITALES - **Parcial**

Primer Cuatrimestre 2024

Ej. 1	Ej. 2	Ej. 3	Ej. 4	Nota

Correctorx:

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 cuatro datos sin signo de un byte cada uno almacenados en el registro `s0` y queremos sumar el valor de los cuatro datos.. Escriba un programa de ensamblador `RISC V` que realice esta operación y almacene el resultado en el registro `a0`.

Ejemplo:

Bits	31 24	23 16	15 8	7 0
s0	0x90	0x1A	0x00	0x02

Con este dato el registro debería valer 0x000000BC.

Ejercicio 2 Implemente la función hanoi 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.

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

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 sistema de gestión de notas mantiene registro de las notas de una clase en un arreglo de datos de 1 byte sin signo. Queremos agregar lógica para determinar la cantidad de estudiantes que obtuvieron un valor mayor a 0xA0, que es el valor con el cual promocionan.

Se cuenta con un arreglo `notas` 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 cuente la cantidad de notas que se encuentran por sobre el valor `0xA0`. Si la cantidad de valores que superan este límite es mayor a la mitad del largo debemos poner un 1 en el registro `a0`, en caso contrario debemos poner un 0.

Ejemplo:

DNI: 45238040

W: 448/23

MONTERO SUAN CRUZ 1

B-

① EJERCICIO 1:

```
ADDI t0, s0, 0
ADDI t1, s0, 0
ADDI t2, s0, 0
ADDI t3, s0, 0
```

GUARDO EN t0, t1, t2, t3 EL VALOR DE S0

t0 SIMBOLIZARA EL BYTE 0 LUEGO DE LIMPIARLO

t1 " " " 1 " " "

t3 " " " 3 " " "

```
//LIMPIO [illegible] t0 PARA QUEDARME CON [illegible] BYTE 0
SLLI t0, t0, 24
SLRI t0, t0, 24

//LIMPIO t1 PARA QUEDARME CON BYTE 1
SLLI t1, t1, 16
SLRI t1, t1, 24

//LIMPIO t2 PARA QUEDARME CON BYTE 2
SLLI t2, t2, 8
SLRI t2, t2, 24

//LIMPIO t3 PARA QUEDARME CON BYTE 3
SRLI t3, t3, 24

//LOS SUMO EN a0
ADD a0, t0, t1
ADD a0, a0, t2
ADD a0, a0, t3

RET
```

RET ES UNA PSEUDO INSTRUCCION PARA DALR X0

EN ESTE CONTEXTO NO ES NECESARIO HACER ESO

NOTA

EXPLICACIÓN DEL CÓDIGO.

① ME COPIO EL VALOR DE s_0 EN t_0, t_1, t_2, t_3

② LIMPIEZA DE t_0

DADO QUE EL BYTE 0 INVOLUCRA LOS BITS DE 0 AL 7

VOY A QUERER QUEDARME CON ESO EN t_0.

⑤ MUEVO TODOS LOS BITS A ~~DERECHA~~ IZQUIERDA Y DADO QUE USO SLLI

ME COMPLETA CON CEROS A IZQ). UNA VEZ QUE MOVÍ A IZQ "24 BITS"

TENGO EL BIT "0" EN LA POSICIÓN 24, EL 1 EN LA 25 ...

Y EL 7 EN LA 31.

⑥ MUEVO EL BYTE A SU LUGAR CORRESPONDIENTE

PARA ESO HAGO SRLI QUE ME COMPLETA ~~LOS [illegible] A BIT 0~~ CON [illegible]

Y DESPLAZO LOS BITS TANTO COMO DESEE. PARA EL PRIMER CASO

TENGO QUE LLEGAR A LOGRAR QUE EL BIT 24 (POST SLLI) ESTÉ

EN EL BIT 0, POR LO TANTO DESPLAZO A DERECHA 24 BITS

③ CASO LIMPIEZA BYTE 3. DADO QUE YA ESTÁ LO MÁS A LA IZQUIERDA

POSIBLE, SI LO DESPLAZO PIERDO INFORMACIÓN, PARA EVITAR ESTO NO

LO DESPLAZO A IZQ, SINO QUE A DERECHA "24 BITS" PARA

LOGRAR QUE EL BIT 24 QUEDE EN EL BIT (0)

④ HAGO LA SUMA BIT A BIT ENTRE CADA UNO DE LOS REGISTROS

Y LA ALMACENO EN a_0.

DNI: 45238040

LW: 448/23 MOUREO SUAN CRUZ

B ② HANOI PSEUDO CÓDIGO.

```
HANOI (M):
IF (M == 1)
  RET 1
ELSE
  TEMP0 = HANOI (M-1)
  TEMP1 = 2 * TEMP0
  TEMP2 = TEMP1 + 1
  RETURN TEMP2
```

REGISTROS: a0 = M
ra
t0 = 1 // PARA GRAFICAR IF

VARIÁVEIS A GUARDAR: ra
EN MEMORIA

CÓDIGO RISC V

COMENTARIO: ASUMO CONTRATO M ≥ 1

```
HANOI: ADDI t0, ZERO, 1      (1)
       BEQ a0, t0, BASE      (2)
       ADDI sp, sp, -4       (3)
       SW ra, 0(sp)          (4)
       ADDI a0, a0, -1       (5)
       CAL HANOI             (6)
       [illegible]           (7)
       ADDI a0, a0, 1        (8)
       LW ra, 0(sp)          (9)
       ADDI sp, sp, 4        (10)
       RET                   (11)

BASE: RET                    (12)
```

DNI: Y5038040
Lu: 448/23

Monzero Svan Cruz

B⁻ ③ [illegible]

3

EXPLICACIÓN:

① Asigno 1 al registro t0 para verificar el IF.
② Verifico si m (a0) es igual a 1, si es igual salto al caso base, sino a la instrucción siguiente [sumo 4 al PC].
③ Busco guardar un espacio en memoria, por eso resto 4 al PC.
④ Guardo el valor de retorno en memoria.
⑤ Resto 1 a a0 para así calcular el hanoi (m − 1).
⑥ Hago un ~~[illegible]~~ salto a hanoi guardando en ra el valor del PC +4 (guarda la sig. instrucción)
⑦ Multiplico a0 por 2 (desplazo 1 bit a izq) ← 2 × hanoi (m−1)
⑧ Sumo 1 a a0 ← 2 × hanoi (m−1) +1
⑨ Cargo el valor de la memoria (un valor ra almacenado anteriormente) en ra
⑩ Restauro / limpio la memoria incrementando en 4
⑪ Hago ret para saltar al valor del ra obtenido con el sw
⑫ Caso base, sé que a0 = 1 ya que caí en el caso base entonces me alcanza con ~~[illegible]~~ volver al lugar que indica el ra con un ret y no cambiar el valor de a1 que ~~[illegible]~~ en el ~~[illegible]~~ pseudocódigo quería retornar 1

✓

NOTA

DNI: Y5238040
LU: 448/23

MONTERO JUAN CRUZ

3

B⁻

③ ~~[illegible]~~

```
.TEXT

LA  a0, NOTAS
LA  a1, LARGO
~~[illegible]~~

LBU a1, 0(a1)   // VALOR DE LARGO SIN EXTENSIÓN DE SIGNO

ADDI t0, a1, 0       // CONTADOR DEL LARGO
ADDI t3, zero, 0     // CONTADOR MAYORES < 0xA0

WHILE: BEQ t0, zero, FIN
       LBU t1, 0(a0)
       ADDI t4, zero, 0xA0
       SLTU t2, t4, t1      // 1 SI t4 < t1 (PARÁMETRO / LA NOTA) SINO 0
       ADD t3, t3, t2       // SUMO EL VALOR AL CONTADOR
       ADDI t0, t0, -1
       ADDI a0, a0, 1
       J WHILE

FIN:   ADDI t0, a1, 0    // ME VUELVO A TRAER EL VALOR DEL LARGO
       SRLI t0, t0, 1
       SLT a0, t0, t3    // 1 SI LARGO/2 < CONTADOR
       RET               MISMO QUE EJERCICIO 1)
```

NOTA

DNI: 45238040
W: 448/23

MONTERO JUAN CRUZ

4

B

⑤ La ventaja de tener un registro constante cero es en parte la facilidad para crear pseudoinstrucciones, las cuales le facilitan al programador la creación de código más claro y conciso. Las ~~[illegible]~~ pseudoinstrucciones son una configuración de las instrucciones de RISC-V con alguna configuración especial en los registros. Por ejemplo si tomamos RET.

Esta instrucción básicamente es una JALR x0, x1, 0 y al mismo tiempo es mucho más fácil de escribir y recordar.

Esta creación de pseudo instrucciones se permiten ya que el registro x0 no permite su escritura y por eso si trato de escribir en el no va a afectar en nada al registro y va a seguir siendo el valor constante "0".

Otro punto de vista es que ~~[illegible]~~ RET está usando el registro x0 para asignar ~~[illegible]~~ un registro de destino al cual no va a poder (ni le interesa) modificar, ya que a RET solo le interesa saltar al valor del PC almacenado en RA.

⑥ Las escrituras, por lo dicho anteriormente, están prohibidas. Es decir si yo trato de pisar el valor de x0 con otro valor simple no va a hacer nada, su valor seguirá constante en 0.

Mientras que la lectura al registro sí están habilitadas, o sea si trato de leer ~~[illegible]~~ el valor de x0 voy a obtener un "0".

ENTIENDO QUE ENTENDÉS LA IDEA, NO ES QUE ESTÉN PROHIBIDAS Y SE GENERE UN ERROR, SINO QUE CUANDO SE TRATE DE ESCRIBIR NO VA A TENER EFECTO ALGUNO.

NOTA

EJERCICIO 3 EXPLICACIÓN:

DADO QUE EL DATO DE NOTAS VIENE EN BYTES ENTONCES:
PRIMERO HAGO UN FETCH DEL 0 BYTE. PARA ESO PRIMERO OBTENGO
LA ADDRESS PARA ASÍ TENER LA DIRECCIÓN AL PRIMER BYTE DEL ARRAY.
LUEGO HAGO LO MISMO CON LARGO Y PARA OBTENER EL LARGO SIN
SU EXTENCIÓN DE SIGNO ENTONCES HAGO LBU CON $s1 = LA ADDRESS DE LARGO.
LUEGO ME CREO DOS CONTADORES, 1 QUE SIRVE EL RECORRIDO EN EL ARRAY
(SIRVE LA CANTIDAD DE ELEMENTOS RESTANTES) Y OTRO QUE CUENTE LA
CANTIDAD DE NUMEROS MAYORES A LA COTA (COTA)

LUEGO ENTRO AL WHILE Y VERIFICO SI EL CONTADOR DE LARGO ES IGUAL
A "0". SI NO ES IGUAL ENTONCES VOY A BUSCAR CARGAR EL BIT DE NOTAS
EN DONDE TENGA LA ADDRESS APUNTANDO (ACA) Y LO HAGO CON LBU PARA EVITAR
LA EXTENCIÓN DE SIGNO. EN LA PRÓXIMA INSTRUCCIÓN CARGO LA COTA A t4
PARA ASÍ CON SLTU PODER VERIFICAR SI LA COTA ES MENOR A LA NOTA,
SI ESTO PASA ~~PONE~~ PONE UN 1 EN t2 ~~LA CANTIDAD~~, SINO UN 0. DESPUES SUMO EL
VALOR AL CONTADOR DE NOTAS MAYORES Y INCREMENTO LA ADDRESS EN 1
(YA QUE ME VOY A ~~QUERER~~ MOVER AL PROXIMO BYTE) Y DECREMENTO
EL CONTADOR DE LARGO EN 1 (PARA ~~SABER~~ SABER QUE YA COMPUTE UN DATO DEL ARRAY)
Y VUELVO AL WHILE.

POR ÚLTIMO LUEGO A FIN UNA VEZ QUE NO ME QUEDEN MAS ELEMENTOS
EN EL "ARRAY NOTAS" (OSEA CUANDO MI t0 SEA 0). AQUÍ VOY A VOLVER
A TRAERME EL VALOR SIN SIGNO ALMACENADO EN a1 DEL LARGO, ~~VOLVER~~ LO VOY A
DIVIDIR POR 2 CON UN CORRIMIENTO DE UN BIT A DER. ~~Y VOY A VOLVER~~
~~AL CORRECTO~~
POR ÚLTIMO COMPARO Y ALMACENO EN a0 EL RESULTADO DE UNA VEZ
SI EL LARGO/2 ES MENOR AL CONTADOR DE NOTAS MAYORES A LA COTA