SISTEMAS DIGITALES - **Parcial**

Primer Cuatrimestre 2024

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

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 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?

(1) #Asumo que los datos están cargados en S0.

```
Li t0  0x0000FFFF   #Máscara parte baja
Li t1  0xFFFF0000   #Máscara parte alta
```

```
AND t2 s0 t0    #t2 es la parte baja de s0
SLLi t2 t2 16   #Acomodo el dato en la parte alta.
                (La parte baja tiene ceros).
MV a0 t2        # a0 ← t2

AND t2 s0 t1    #t2 es la parte alta de s0
SRLi t2 t2 16   #Acomodo el dato en la parte baja.
                (La parte alta tiene ceros).
add a0 a0 t2    # a0 ← a0+t2
```

#El resultado queda en a0 como se pide.

#Usamos SRLi en vez de SRAi para asegurarnos que la parte alta de t2 tenga 0s de cero en la parte alta.

Resolvemos el ejemplo: s0 ⇝ 0x1A90 0200

t2 ← 0x0000 0200 = 0x1A90 0200 ∧ 0x0000 FFFF

t2 ← 0x0200 0000 = shift 16 BITS ← lógico a 0x0000 0200

a0 ← 0x0200 0000

t2 ← 0x1A90 0000 = 0x1A90 0200 ∧ 0xFFFF 0000

t2 ← 0x0000 1A90 = shift 16 BITS → lógico a 0x1A90 0000

a0 ← 0x0200 1A90 = 0x0200 0000 + 0x0000 1A90

Respuesta.

B

② Primero escribimos la función en PSEUDOCÓDIGO.

```
def rec(n:Int){
    if (n==1) then return 1        → debería ser 0
    else return 2*n+rec(n-1)
}
```

Implementamos la función en el lenguaje ensamblador RISC V.

⋮ Código anterior.

#Definición de la función. #Asumo que la llamo con a0 cargado con el argumento n (positivo).

```
rec:
    Li t1 1          # t1 lo uso como cte 1 para saber
                     si se cumple el caso base.
Recursión:
    BEq a0 t1 cBase  #¿n==1?
    addi sp sp -16   # Pido memoria (Solo necesito 8 BYTE por
                     la convención dice que el
    sw a0 0(sp)      #Guardo el n   sp se mueve en
    sw ra 4(sp)      #Guardo el ra  multiplos de 16)
    addi a0 a0 -1    # n-1 en a0
    jal ra Recursion #a0 ahora vale rec(n-1)
    Lw t0 0(sp)      # t0 es n
    sLLi t0 t0 1     #Multiplico por 2, t0 es 2*n.
    add a0 a0 t0     #a0 es rec(n) = 2*n (t0) + rec(n-1) (a0)
    Lw ra 4(sp)      #Recupero el ra
    addi sp sp 16    #Libero memoria.
    jr ra

cBase:
    Li a0 0          # Luego que la función termina,
    jr ra            el resultado queda guardado en
                     a0 (Siguiendo la convención).
```

OBS: • Los registros que modifica son $a0$, ta, tY que no los guarda la función llamada. (Cumple la convención).

• Otro registro que modifica es el SP, pero nos aseguramos que quede en la misma posición. (Cada vez que entra en la recursión resta 16 (pide memoria 16 BYTES) y cada vez que sale de la recursión suma 16 (libera memoria) 16 BYTE)

• El registro ra se fue guardando en memoria para saber hacia donde volver durante la función recursiva.

Los n's los guarda en $0(SP)$

y los ra's los guarda en $4(SP)$

B

③ .data
puntajes: .byte 0x07 0xF1 0xB1 0x07
largo: .byte 4

.text
LA a0 puntajes #a0 dirección de inicio del arreglo.
LB a1 largo #a1 es el largo (Ojo que LB extiende el signo)
jal ra deteccionTrompo #Llama a la función.
Li a7 10
eCall #Termina programa.

DeteccionTrompo:
Li t0 0x0F0 #t0 es el umbral, puntaje máximo.
CICLO:
BEQZ a1 todoHonesto
LBU t1 0(a0) #t1 dato actual (SIN SIGNO)
BLTU t0 t1 trompo #Si P. máximo < dato actual → Hay trompo
#Siguiente
addi a0 a0 1 #Corrido que el angulo empieza en el siguiente byte.
addi a1 a1 -1 #Reduzco en 1 la longitud
j ciclo

trompo:
Li a0 1 #Resultado 1. #El resultado, con fido el enunciado, se guarda en a0.
ret

todoHonesto:
Li a0 0 #Resultado 0
ret

④ Positional Independent Code (PIC) significa que el comportamiento del código es independiente a la posición en la que este ubicado en la memoria.

Los etiquetas en los saltos (tanto condicionales como incondicionales) son resueltos como inmediatos relativos a la posición que debe llegar el PC desde esa posición actual.

Al ser relativos, no afecta la posición en la que empieza el programa en la memoria.

Informalmente, "no le indican a donde ir sino como moverse" (hacia que dirección y que tanto).

Un ejemplo en donde lo vimos fue al cambiar la ejecución del programa del ejercicio 3 cambiando su comportamiento cuando empezaba a alocar el programa a partir de 0x0 y a partir de 0x8.

Si usamos inmediatos en vez de etiquetas para las instrucciones de tipo J y B:
Interpretamos la misma de 2 maneras distintas, tenemos ambas interpretaciones.
INTerpretación 1: Si con el inmediato le decimos a que posición de la memoria tiene que ir el PC:
Tenemos el problema si se guarda el código en otra posición de memoria. No es PIC.

+ Deberá contar PSEUDO-instrucciones
(descompone)

Si en el inmediato escribimos

Interpretación 2: [illegible] el relativo:

El programador tendrá problemas si necesita escribir código [illegible] entre etiquetas, deberá asegurarse [illegible] de volver a acordar por inmediato.

(Sigue siendo necesario descomponer grandes instrucciones).

Resolver etiquetas no debería ser responsabilidad del programador.