Nro. ord.	Apellido y nombre Lista Melanie Carolina	L.U. 516/21

ORGANIZACIÓN DEL COMPUTADOR I - **Parcial**

2023 1C

Ej.1	Ej.2	Ej.3	Ej.4	Ej.5	Nota

Corrector:

Aclaraciones

- Anote apellido, nombre, LU en *todos* los archivos entregados.
- El parcial es domiciliario y todos los ejercicios deben estar aprobados para que el parcial se considere aprobado. Hay dos fechas de entrega, en ambos casos el conjunto de ejercicios a entregar es el mismo. En la primera instancia deberán defender su trabajo frente a su tutorx, quien les ayudará también a encaminar el trabajo de los ejercicios pendientes, si los hubiera.
- El link de entrega es: `https://forms.gle/9yA4s2PHfBU4uXvM6`. Ante cualquier problema pueden comunicarse con la lista docente o preferentemente con el/la corrector/a.
- La fecha límite de entrega es el jueves 29 de Junio a las 17:00. El coloquio será el martes 4 de Julio en el horario de cursada de los jueves (TM: 9 a 13hs - TT: 17 a 21hs) de **forma presencial** en un aula que figura en el calendario.
- Todas las respuestas deben estar correctamente justificadas.

Introducción

Este parcial está dividido en cuatro ejercicios de implementación de cóodigo ensamblador para `RISCV32I`. Uds van a recibir un archivo con un esqueleto de código que deben completar, pueden utilizar el simulador `RIPES` para probar su programa. Pueden descargarlo en `https://github.com/mortbopet/Ripes`. Toda la información necesaria está disponible en la `Guía Práctica de RISC-V` que se puede acceder libremente en:
`http://riscvbook.com/spanish/guia-practica-de-risc-v-1.0.5.pdf`.

Esperamos que entreguen el archivo con la implementación y otro donde expliquen cómo resolvieron los ejercicios.

Ejercicios

Ejercicio 1

Escribir la función `restar_limpiando_impares` que devuelve el resultado de limpiar los bits impares de a, restar el resultado a b y luego esto a *accum*.

```
int32_t restar_limpiando_impares(int32_t accum, int32_t a, int32_t b){
  a = a & 0x55555555;
  return accum + a - b;
}
```

Ejercicio 2

Escribir la función `restar_limpiando_pares` que devuelve el resultado de limpiar los bits pares de a, restar el resultado a b y luego esto a *accum*.

```
int32_t restar_limpiando_pares(int32_t accum, int32_t a, int32_t b){
  a = a & 0xaaaaaaaa;
  return accum + a - b;
}
```

Ejercicio 3

Escribir la función `posicion_par` que devuelve 1 si *index* es par, 0 en caso contario.

```
int32_t posicion_impar(int32_t a, int32_t index, int32_t length){
  return (index & 0x1) != 0x1;
}
```

Ejercicio 4

Escribir la función `numero_negativo` que devuelve 1 si a es negativo, 0 en caso contario,

```
int32_t numero_negativo(int32_t a, int32_t index, int32_t length){
  return a < 0;
}
```

Justificacion

Ejercicio 1)
Para poder limpiar los bits impares de "a" debo usar alguna operación que permita acceder a los bits del parámetro.
La operación AND es una operación que realiza un "y" lógico bit a bit. Por lo tanto, si quiero dejar los bits impares en 0, entonces debo encontrar un valor que sea de la forma 1010 1010 (...) 1010 con 1 en las posiciones pares y 0 en la posiciones impares. Las palabras en risc-V son de 32 bits, por lo tanto el valor buscado es de la forma 0xAAAAAAAA ya que cada digito en hexa es de 4 bits y A=1010 y por lo tanto obtenemos un valor de 8 dígitos ya que 8*4=32.
Luego una vez limpiados los bits impares, se pueden realizar las operaciones suma y resta con las operaciones Add y Sub respectivamente. Utilizo registros que son temporales por convención porque son valores que uso una única vez para realizar las operaciones. También lo uso para preservar los valores pasados por parámetro. Lo único que modifico es el registro a0 ya que por convención, ese registro también se usa para los valores de retorno.

Ejercicio 2)
La idea es la misma a implementar que el ejercicio 1. Pero, en este caso para cceder a los valores pares necesitaria un valor de la forma 0101 0101 (...) 0101. Las palabras en risc-V son de 32 bits, por lo tanto el valor buscado es de la forma 0x55555555 ya que cada digito en hexa es de 4 bits y 5 = 0101 y por lo tanto obtenemos un valor de 8 digitos ya que 8*4 = 32 bits.

Ejercicio 3)
Para poder saber si un numero es impar o par puedo aprovechar el hecho de que los valores estan expresados en bits. Un numero es impar si su valor en bits tiene un 1 en el bit menos significativo, por ejemplo:

10= A = 1010 >> es par pues 2^3 + 2^1 = 10
11 = B = 1011 >> es impar pues 2^3 + 2^1 + 2^0 = 8+2+1=11

Para poder acceder al bit menos significativo es utiliza la misma logica que los ejercicios anteriorres. El "y" lógico bit a bit es la que me permite limpiar todas las posiciones menos la ultima. Es decir necesito un valor que tenga 31 ceros y 1 uno al final = 0x00000001
Una vez conocido el valor del ultimo bit, es cuestión de saber si es 0 o 1. Para eso tenemos una operación "BEQZ"= Branch if Equal to Zero. Si el bit es igual a 0 implica que el valor es par, en caso contrario es impar.

Ejercico 4)
Para saber si un numero es negativo solo necesito saber si es menor a cero. Risc-V permite acceder a esta informacion con la operacion BLTZ=Branch if Less to Zero.