

Nro. ord.	Apellido y nombre	L.U.

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 código ensamblador para RISC-V32I. Uds van a recibir un archivo con un esqueleto de código que deben completar, pueden utilizar el simulador RIPS 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 $accum$.

```

1 int32_t restar_limpiando_impares(int32_t accum, int32_t a, int32_t b){
2     a = a & 0x55555555;
3     return accum + a - b;
4 }
```

Ejercicio 2

Escribir la función `sumar_extender` que devuelve el resto de extender el signo de a , suponiendo que es un dato de 16 bits, sumar el valor nuevo a b y sumar esto, a su vez, a $accum$. Si a o b son cero, devuelve cero, si b es negativo, también devuelve cero.

```

1 int32_t sumar_extender(int32_t accum, int32_t a, int32_t b){
2     a = ((a & 0x8000) != 0x0)? a | 0xFFFF0000 : a & ~0xFFFF0000;
3     return accum + a + b;
4 }
```

Ejercicio 3

Escribir la función `posicion_impar` que devuelve 1 si *index* es impar, 0 en caso contrario.

```
1 int32_t posicion_impar(int32_t a, int32_t index, int32_t length){  
2     return (index & 0x1) == 0x1;  
3 }
```

Ejercicio 4

Escribir la función `numero_par` que devuelve 1 si *a* es par, 0 en caso contrario.

```
1 int32_t numero_par(int32_t a, int32_t index, int32_t length){  
2     return (a & 0x1) != 0x1;  
3 }
```