Ejercicio 1

Muchos ISAs utilizan un campo como origen en algunas instrucciones y como destino en otras, **forzando el agregar hardware extra en sitios potencialmente críticos** (en tiempo) para seleccionar el campo correcto.

El problema con esto es que deberíamos tener dos formas de decodificar las mismas operaciones.

Una para cada posición distinta de los registros, forzando agregar hardware extra en el decoder.

Ejercicio 2

```
#este es un comentario
 .text #comenzamos con esta directiva

#completar

main:    add x11, x0, x0

         add x11, x10, x0    #acumulador sumas desplazamientos

desplazar:    srli x10, x10, 1

              add x11, x11, x10

              bne x0, x10, -8 #el registro x0 siempre tiene
                                                          el valor 0

#finalizamos con estas líneas
fin : addi x17 , x0 , 0x11
ecall
```

posiciones de memoria de las instrucciones:

add 0
add 4
srli 8
add 12
bne 16

registros utilizados:

x10, x11 son únicamente los mencionados en el enunciado.

decodificación de las instrucciones:

```
sw x10, 20(x10)          0000000 01010 01010 010 10100 0100011

lw x11, 20(x10)          000000010100 01010 010 01011 0000011

srli x10, x10, 1         0000000 00001 01010 101 01010 0010011

add x11, x11, x10        0000000 01010 01011 000 01011 0110011

bne x0, x10, -8          1111111 01010 00000 001 10001 1100011
```

Ejercicio 3

¿Cuáles son los objetivos no funcionales que guían el diseño del set de instrucciones de RISC-V? ¿Cómo impactan en su diseño?

Los objetivos no funcionales que guían el diseño del set de instrucciones de RISC-V son:

- Debe acoplarse a todos los procesadores.
- Debe funcionar bien con una amplia variedad de paquetes de software y lenguajes de programación populares.
- Debe poder implementarse en todo tipo de tecnologías.
- Debe ser eficiente para todo tipo de micro-arquitecturas.
- Debe permitir un alto grado de especialización para utilizarse como base en aceleradores personalizados, los cuales cobran mayor importancia ahora que la Ley de Moore comienza a perder fuerza.
- Debe ser estable, implicando que el ISA base no debe cambiar. Aún más importante, no debe descontinuarse.
- El objetivo es mantener compatibilidad binaria para que los programas ya compilados y en formato binario de décadas anteriores, aún puedan funcionar en los procesadores más recientes.
- El núcleo fundamental del ISA es llamado RV32I, el cual ejecuta un stack de software completo. RV32I está congelado y nunca cambiará, lo cual provee un objetivo estable para desarrolladores de compiladores, sistemas operativos y programadores de lenguaje ensamblador.
- RISC-V no tiene necesidad de agregar instrucciones por cuestiones de mercadeo.

Principios básicos y los sacrificios/compromisos que debe tomar un arquitecto de computadoras al momento de diseñar un ISA.

- costo: el costo es muy sensible al tamaño del die
- simplicidad: dada la correlación costo-complejidad, los arquitectos prefieren un ISA simple para reducir el tamaño del die. La simplicidad también reduce el tiempo requerido para diseñar y validar, lo cual es un porcentaje importante del costo de desarrollo de un chip.
- rendimiento: la simplicidad también reduce el costo de la documentación y la dificultad de hacer que los clientes entiendan cómo usar el ISA. A excepción de chips muy pequeños utilizados para aplicaciones embebidas, los arquitectos se preocupan tanto por rendimiento como costo. Las instrucciones simples son además las más populares, así que un ISA más simple puede ganar en todas las métricas
- aislamiento de arquitectura e implementación: A pesar que los arquitectos no deberían agregar funciones que ayudan a una implementación específica, tampoco deberían agregar funciones que afecten algunas implementaciones.
- espacio para crecer: el único camino para incrementar significativamente el rendimiento es agregar instrucciones para tareas específicas. Esto implica que es fundamental reservar espacio en el opcode para dichas instrucciones.

- ➢ tamaño del programa: Entre más pequeño el programa, menor el área del chip que ocupará la memoria, lo cual puede ser un costo significativo para sistemas embebidos. Programas más pequeños además implican menos instruction cache misses, lo cual ahorra energía porque accesos a memoria DRAM externa consumen mucho más energía que accesos a SRAM en el chip.
- ➢ facilidad de programar / compilar / linkear: Dado que el acceso de datos en registros es mucho más rápido que en memoria, es crucial que los compiladores hagan una buena asignación de registros. Dicha tarea es más fácil entre más registros se tenga. Más registros hacen la tarea más fácil para programadores de ensamblador y compiladores. Instrucciones complejas y operandos en memoria incrementan la dificultad para los diseñadores de procesadores en proveer estimaciones de desempeño.

En resumen, el objetivo de los arquitectos de RISC-V es que sea eficaz para todos los dispositivos de cómputo, desde los más pequeños hasta los más rápidos. Siguiendo el consejo de von Neumann de hace más de 70 años, este ISA enfatiza simplicidad para mantener el costo bajo y muchos registros y velocidad de ejecución transparente para ayudar a compiladores y programadores a resolver problemas importantes de manera eficiente.

Ejercicio 4

¿Qué problema puede introducir habilitar interrupciones cuando se atiende una interrupción en OrgaSmallI?

¿Qué pasa cuando interrumpen al CPU?

En el caso de un procesador OrgaSmallI, si el dispositivo de E/S activa la señal de interrupción y el flag I vale 1, termina de ejecutar la instrucción en curso y realiza atómicamente los siguientes pasos:

Coloca `[0xFF] = PC`

Coloca `I=0` para evitar que el procesador vuelva a interrumpirse

Coloca `PC = [0x00]`

Activa la señal INTA para indicarle al dispositivo que atenderá su pedido

Luego, comienza a ejecutarse la rutina de atención de la interrupción propiamente dicha.

Guardando el PC de la instrucción siguiente a ejecutar luego de la rai en la posición de memoria 0xFF y lo que hay en la posición de memoria 0x00 en el PC para que se ejecute la rai.

Ahora suponiendo que no se coloca `I=0`, en el fetch sólo se ejecutarán los pasos de:

```
Coloca [0xFF] = PC   # guarda la siguiente instrucción a ejecutar
Coloca PC = [0x00]   # ejecuta la rai
```

Activa la señal INTA para indicarle al dispositivo que atenderá su pedido

Luego, comienza a ejecutarse la rutina de atención de la interrupción int2 propiamente dicha.

Si mientras se ejecuta la rai se recibe una interrupción int2, se ejecutarán los siguientes pasos:

```
Coloca [0xFF] = PC   # el PC está en 0x00 porque se está ejecutando una rai
Coloca PC = [0x00]   # ejecuta la rai
```

Activa la señal INTA para indicarle al dispositivo que atenderá su pedido

Luego, comienza a ejecutarse la rutina de atención de la interrupción int2 propiamente dicha.

Por lo tanto, la rutina de atención a la interrupción anterior no terminaría nunca de ejecutarse porque en la dirección en donde se guarda la siguiente instrucción a ejecutar se pisó a 0x00 y no se podría salir de la rai.

Ejercicio 5

Escriba el microprograma que permita sumar el contenido del valor que se encuentra en la dirección referida por el registro indicado en el **operando X del decoder** consigo mismo.

```
;[Rx] <- [Rx] + [Rx]
```

#código de operación de la microinstrucción

```
xxxxx: ; SUMA_DIR
     RB_enOut RB_selectIndexOut=0 mem_op=010
     RB_enOut RB_SelectIndexOut=0 mem_Op=001
     mem_op=100
     reset_microOp
```