TRABAJO PRÁCTICO INTEGRADOR:
Gestión de excepciones en un RISC-V

Alumnos: Escudero, Sofía
Pighin, Nael
Reus, Martín

Docentes: Leonardo Giovanini
Eugenio Juan Padula

AÑO: 2022

Resumen

En el siguiente informe se llevará a cabo el desarrollo de un sistema que involucra tanto hardware como software, cuyo objetivo principal consiste en procesar un bucle de incremento de un contador. Mientras se realiza esta tarea, el mismo debe ser capaz de atender una entrada externa proveniente de un pulsador, el cual puede ser activado aleatoriamente y sin seguir ningún patrón. Cada vez que se presione el pulsador se debe cambiar la salida para encender un LED.

El hardware estará desarrollado mediante código VHDL que implementará un microprocesador RISC-V, desarrollado previamente en la asignatura, el cual será modificado para ejecutar nuevas instrucciones que no estaban soportadas y atender una interrupción de un periférico. Por otra parte, el módulo de software será realizado mediante instrucciones en código ensamblador, simulado en el programa RARS, las cuales serán las encargadas de representar las tareas que debe realizar el procesador para procesar el bucle del contador y para gestionar el apagado y encendido del led. Por último, para visualizar y comprender el funcionamiento de la lógica de este proyecto, se llevará a cabo un testbench donde podrán visualizarse entradas y salidas del procesador, así como también señales internas de interés.

Palabras clave— RISC-V, interrupción, instrucciones, assembler.

Introducción

El RISC-V es una arquitectura de conjunto de instrucciones (ISA) libre y abierto. Es de tipo RISC (conjunto de instrucciones reducido), esto es, sólo implementa las instrucciones más utilizadas para optimizar el desempeño global.

Previo al presente trabajo, se construyó en VHDL un procesador RISC-V monociclo que pudiera implementar instrucciones lw, sw, tipo R, beq, tipo I (addi) y jal.

En este informe, se detallan las implementaciones necesarias a este RISC-V para que procese un bucle de incremento de un contador y atienda la entrada externa y aleatoria de un pulsador, el cual cambiará la salida de un led. El led en cuestión lee la posición 0x00000000 de la memoria de datos para detectar su estado de encendido o apagado, representados por los valores 0x00000001 y 0x00000000 respectivamente.

Arquitectura del Hardware

El RISC-V implementado durante el cursado tiene la siguiente arquitectura:

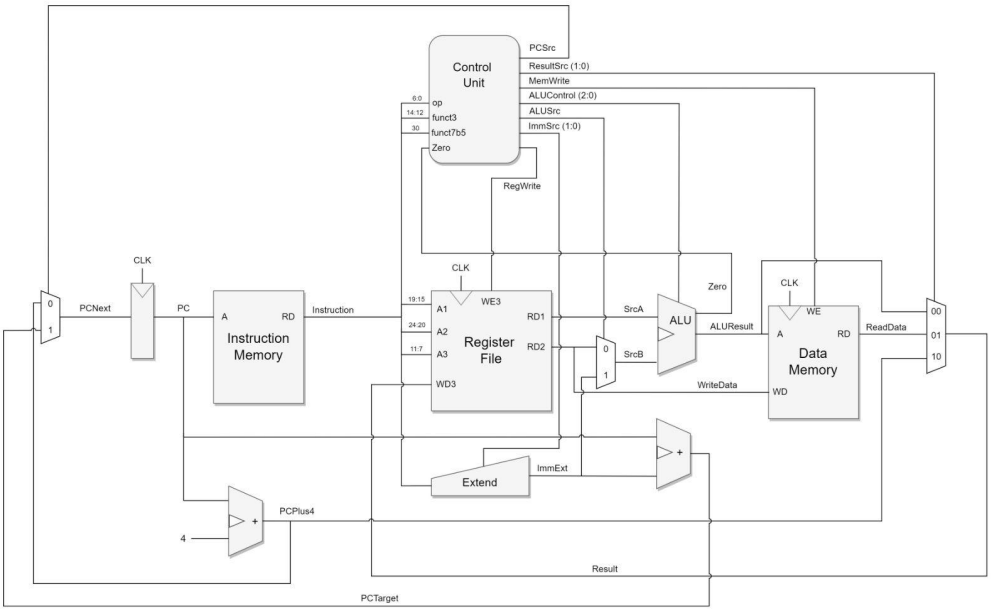

Fig. 1: Arquitectura del procesador monociclo implementado en clases.

Contador de Programa (Program Counter): Tiene como entrada la dirección a la próxima instrucción a ejecutar PCNext y, como salida, la dirección de la instrucción PC que se ejecutará en este ciclo de reloj.

Memoria de Instrucciones (Instruction Memory): Toma la dirección de la instrucción a ejecutar PC y lee la instrucción para pasarsela al archivo de registros.

Banco de registros (Register File): Contiene todos los registros. Tiene dos puertos de lectura y uno de escritura. El banco genera el contenido de los registros correspondientes a las entradas de lectura (A1, A2) en las salidas (RD1, RD2).

Generador de campo inmediato (Extend): Circuito combinacional que genera el campo de dato inmediato. Reorganiza la información del campo inmediato, de acuerdo con el código de operación, para ser utilizada por el procesador para ejecutar la instrucción correspondiente.

Unidad Aritmético Lógica (ALU): circuito digital que calcula operaciones aritméticas (suma, resta, multiplicación, etc.), lógicas (AND, OR, NOT, XOR) y comparaciones para los saltos condicionales entre los operandos. Sus entradas son los operadores (SrcA, SrcB) y la operación a realizar (señal proveniente de la unidad de control). La salida es el resultado (ALUResult) y las banderas de condición.

Memoria de Datos (Data Memory): Región de la memoria del sistema donde se alojan los datos utilizados por el programa (datos dinámicos y estáticos, pila). Sus entradas son la dirección de memoria accedida (A), los datos que se escriben (WD), la señal de escritura (WE) y el reloj del sistema (CLK), la salida es la información (ReadData) que se halla en dicha dirección.

Unidad de Control (Control Unit): Se compone en dos bloques, el decodificador de la ALU y el decodificador principal. Controla el flujo de datos a través del procesador, generando las señales necesarias para operar el camino de datos. Es una máquina de estados finitos cuyas tareas son leer, decodificar, ejecutar las instrucciones y almacenar los resultados.

Partiendo de esta arquitectura como base se realizaron modificaciones para implementar las instrucciones *jalr* y *xori*, necesarias para ejecutar la instrucción encargada de atender la excepción generada por el pulsador que debe activar el led.

La instrucción *jalr* (jump and link register) recupera una dirección almacenada en un registro (preferentemente en el registro ***x1***) y carga en el Program Counter la dirección sumada a un desplazamiento (offset) a definir, permitiendo realizar un salto. Se utiliza generalmente para retornar desde una subrutina al curso normal de instrucciones. A su vez también guarda en un registro el valor actual del Program Counter antes de realizar el salto. Esta instrucción puede utilizarse como complemento para la instrucción *jal*, ya que utilizando *jal* puede realizarse un salto desde el curso de normal de instrucciones hacia una subrutina guardando el Program Counter en un registro, y luego de ejecutar la subrutina se carga la dirección almacenada en el registro para retornar con la instrucción *jalr*.

Para su implementación, se agregó su código de instrucción 1100111 al Main Decoder de la Control Unit. Se decodifica de manera similar a la instrucción jal pero debe activar no solo la señal de salto Jump, sino también una señal extra Reg, la cual indica que la dirección de PC tomada como base para calcular el PCNext se encuentra un registro y debe ser leída. Otra diferencia con la instrucción *jal* es que debe enviarse la señal de activación al generador de inmediatos para extender el valor contenido en la instrucción y llevarlo a un valor de 32 bits.

La instrucción *xori* (exclusive-or immediate) calcula el OR exclusivo a nivel de bits del campo inmediato y un registro fuente rs1, y luego guarda el resultado en otro registro rd. Para implementarla no hizo falta modificar el Main Decoder porque es una instrucción de tipo I pero sí fue necesario añadir el código funct3 de la instrucción a la lista de posibles operaciones en el ALU Decoder, para que este pueda generar como salida el código de operación del XOR que luego se envía a la ALU.

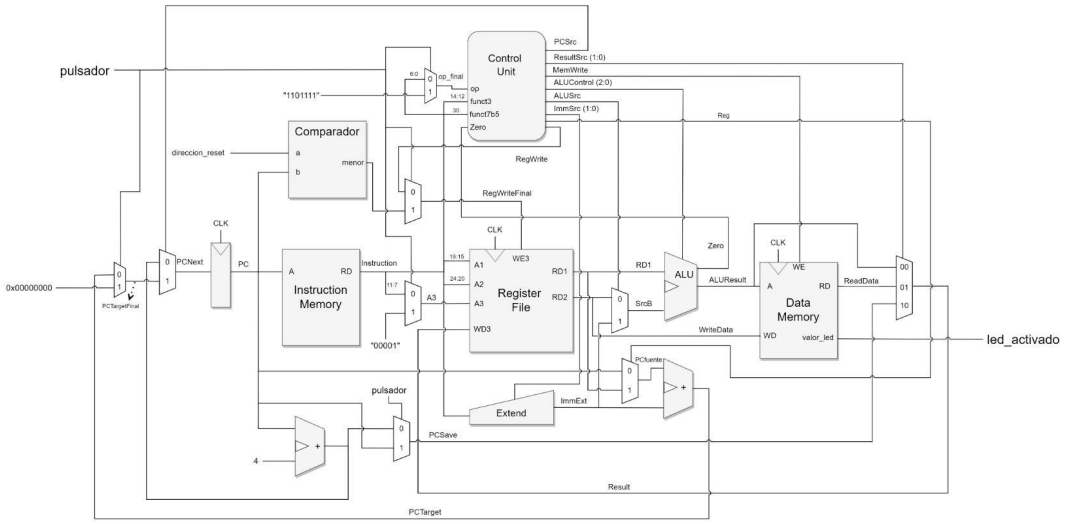

Fig. 2: Arquitectura del procesador monociclo modificada.

Arquitectura del Software

La resolución en software del problema planteado por este trabajo, en código ensamblador del RISC-V, está compuesta por dos programas. En primer lugar, el programa que procesa el contador (Fig. 3) define que el registro donde será guardado el valor del contador es ***x8***, inicializado en 0, y el registro con el cual se comparará para determinar si se debe terminar de contar es ***x9***. En nuestro caso se contará hasta 15. Con esto preparado comienza un bucle sencillo, donde se compara el valor del registro ***x8*** con el de ***x9***, se salta al final del programa si ambos valores son iguales, o se continúa con el bucle si no lo son. En el último caso lo que sigue es sumar el inmediato 1 al valor del registro ***x8*** y saltar nuevamente a la comparación entre registro.

El segundo programa (Fig. 4), encargado de cambiar el estado de activación del led, tiene en cuenta que el puerto al que accede el mismo se encuentra en la dirección de memoria de datos 0x00000000. Sabiendo esto, el programa carga en un registro el valor disponible en esa dirección en el registro ***x6***, invierte el primer bit de 0 a 1 o de 1 a 0 (para encender el led o apagarlo, respectivamente) y finalmente vuelve a guardar el valor en memoria, donde será leído por el led. La inversión del primer bit se lleva a cabo haciendo la operación xor entre el inmediato 1 y el valor almacenado en el registro ***x6***. En último lugar se realiza el salto a la dirección del PC guardada en ***x1*** para poder continuar con el curso normal de instrucciones del contador.

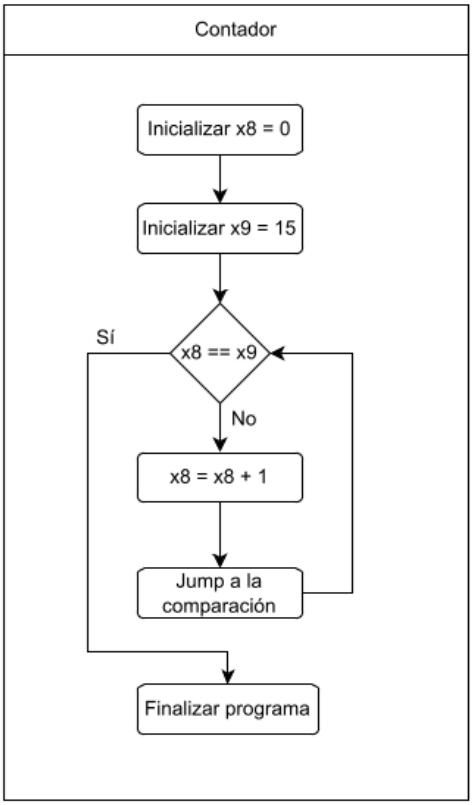

Fig. 3: Diagrama de flujo del programa del contador

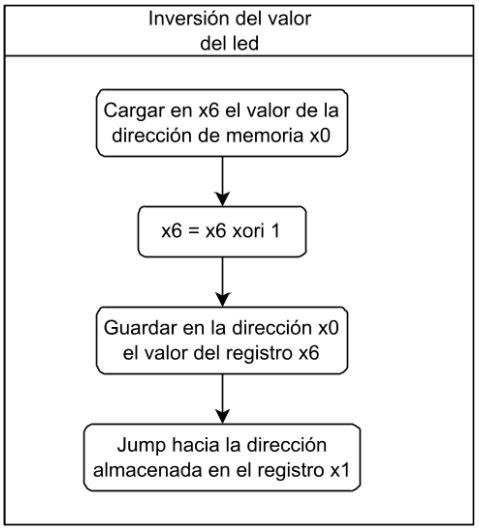

Fig. 4: Diagrama de flujo del programa que gestiona la interrupción generada por el pulsador del led

Implementación de gestión de excepciones

El proceso de gestionar una interrupción producida por un periférico consiste en preguntar en cada ciclo si se ha detectado una interrupción y, si lo ha hecho, guardar el PC en un registro, descubrir qué periférico la ha producido, saltar hacia la dirección de la memoria que corresponda a la interrupción de tal periférico y ejecutar sus instrucciones para finalmente retornar a la dirección del PC almacenada y continuar con la ejecución del programa principal.

Con el fin de gestionar la interrupción del pulsador, representado como una señal de entrada llamada *pulsador*, en primer lugar se definió una sección de la memoria de instrucciones donde se almacenarán las aquellas que gestionan las excepciones, que comienzan desde la dirección 0x00000000 de la memoria. A su vez se cambió la dirección de reset para que quede acorde a la nueva dirección de inicio de las instrucciones de programa.

Para lograrlo se implementaron nuevos multiplexores, cuya señal de decisión es la señal *pulsador*, y que se encargan, en el caso que se haya producido una interrupción con *pulsador* = '1', de fijar las demás señales necesarias para guardar el PC actual y saltar a la dirección 0x00000000. Al fijar las señales que de otro modo dependerían de la instrucción decodificada, se corta la ejecución de la instrucción en ese ciclo, permitiendo así volver más tarde a la misma instrucción sin que se haya ejecutado dos veces. Específicamente, los nuevos multiplexores "transforman" cualquier instrucción en una instrucción *jal* especial, que guarda el PC actual (no el siguiente) en el registro ***x1*** y salta a la dirección 0x00000000 de la memoria de instrucciones. Esto se logra fijando el código de operación a "1101111", el registro donde guardar a ***x1*** y la dirección de salto a la mencionada anteriormente. Al fijar el código de operación automáticamente la unidad de control se encarga de generar las señales necesarias restantes (para el banco de registros, la memoria de datos, la ALU y los otros multiplexores) sin necesitar ninguna modificación adicional.

Una de las consideraciones que se ha definido es que el proceso de gestionar la interrupción se activa cuando el pulsador está activo, pero continúa y finaliza una vez que se desactiva la señal del pulsador. Esto significa que si el pulsador se mantiene activado durante varios ciclos de reloj, entonces el programa "saltará" hacia la dirección de memoria de la interrupción durante esa cantidad

de ciclos. Un inconveniente que se presenta al momento de probar el funcionamiento de estas modificaciones por lo mencionado anteriormente, ya que si el pulsador se mantiene durante varios ciclos, entonces el PC actual se guardará durante varios ciclos, pero en el primer ciclo el PC actual se cambia a la dirección 0x00000000, por lo que se terminará guardando esa dirección y no se podrá volver al curso normal de instrucciones del contador. Para resolver esto, se implementó un nuevo componente, un Comparador, que se encarga de comparar la dirección actual del PC con la dirección de reset de instrucciones aprovechando que las instrucciones de interrupción se encuentran en una dirección menor que las del programa principal en la memoria. Esto permite decidir que, si la dirección actual del PC es menor a la dirección de reset (es decir, si el procesador se encuentra ejecutando las instrucciones de la interrupción) entonces no se debe guardar el PC actual en un registro, porque de hacerlo se caería en la situación de nunca poder volver al programa principal.

CONCLUSIONES

En este informe se ha presentado una implementación para la gestión de excepciones en un RISC-V monociclo, concretamente con el objetivo de manejar señales aleatorias de un pulsador para encender y apagar un LED. Cabe destacar que se podría implementar una solución por sondeo de puertos para obtener los mismos resultados, pero no es objetivo de la consigna.

Gracias a la integración del módulo de software con el módulo de hardware, se logró comprender cuestiones prácticas relacionadas con la teoría aprendida en clases, sobre todo en lo referido a interrupciones y cómo manejar las mismas en un modelo RISC-V.