Desarrollo de un Procesador RISC-V con Gestión de Excepciones e Interrupciones

Axel A. Rueda Gimenez - axelcuer@gmail.com
Universidad Nacional del Litoral - Facultad de Ingeniería y Ciencias Hídricas

Resumen—Este documento describe el desarrollo e implementación de un procesador RISC-V de 32 bits (RV32I) en una arquitectura monociclo con capacidad para gestionar excepciones e interrupciones. El trabajo se enmarca en el contexto de la asignatura Organización de las Computadoras. El objetivo principal fue demostrar el correcto funcionamiento de estas características mediante su implementación en hardware y software. Este proyecto proporciona una base para futuras extensiones.

Palabras clave—RISC-V, gestión de excepciones, interrupciones, registros CSR, arquitectura

I. Introducción

Este informe describe el desarrollo e implementación de un procesador RISC-V de 32 bits (RV32I) con capacidad para gestionar excepciones e interrupciones. El trabajo se enmarca en el contexto de la asignatura Organización de las Computadoras, donde se propuso extender un procesador RISC-V básico, desarrollado durante el cursado, para incorporar funcionalidades avanzadas como la detección y manejo de excepciones e interrupciones. El objetivo principal es demostrar el correcto funcionamiento de estas características mediante su implementación en hardware y validación mediante test bench.

El procesador base implementa un subconjunto de instrucciones del conjunto RV32I, incluyendo operaciones de tipo *R* (aritméticas y lógicas), tipo *I* (carga y operaciones inmediatas), así como instrucciones de acceso a memoria (*lw, sw*), operaciones de tipo *B* (*BEQ*) y de tipo *J*. Se omitió la implementación de operaciones de tipo *U*.

Para lograr la atención a eventos como excepciones e interrupciones, se implementaron módulos y señales de control para gestionar los registros de control y estado (CSR), así como un módulo que detecta y clasifica estos eventos. Estos módulos se integraron en la arquitectura del procesador, permitiendo que este pueda pausar la ejecución, guardar el estado actual y transferir el control a una rutina de manejo de excepciones o interrupciones. Esto permite al procesador responder de manera adecuada a eventos generados por el software (como ebreak) o hardware (como la pulsación de un botón).

El sistema completo se validó mediante la ejecución de un programa contador en lazo cerrado, donde se probaron dos escenarios principales: la generación de una excepción mediante un breakpoint y la generación de una interrupción externa mediante un pulsador. En ambos casos, se verificó que el procesador respondiera de manera correcta, pausando la ejecución y retomándola después de un intervalo de tiempo predefinido.

Este informe detalla los pasos seguidos para el desarrollo del procesador RV32I con circuitos digitales, la implementación del gestor de excepciones e interrupciones, y los resultados obtenidos.

II. Arquitectura del Procesador Base

La arquitectura RISC-V proporciona un esqueleto básico y módulos de expansión para desarrollar procesadores, desacoplando la arquitectura de las implementaciones hardware específicas. Esto brinda libertad a los diseñadores de microprocesadores para crear sus propios diseños dentro del marco de definición de la arquitectura, lo cual permitió que se tomen ciertas decisiones de diseño en esta propuesta de solución.

El procesador base del cual parte este proyecto está definido por el ISA-RV32I con las siguientes restricciones:

- Modo máquina como único modo de operación soportado.
- Se omitieron inicialmente las instrucciones de operación de los registros de control y estado (CSR).
- El procesador utiliza 32 registros de propósito general con una arquitectura Load/Store, y los modos de direccionamiento están limitados a palabras de 32 bits alineadas de a 4.
- Se empleó una microarquitectura monociclo para implementar el procesador, utilizando una configuración Harvard.

En nuestro modelo base del procesador, nos basamos en tres bloques principales:

- *Camino de datos* (*datapath*): Es la ruta que recorren las instrucciones, operandos y resultados.
- *Unidad de control* (*control unit*): Es el camino que siguen las señales de control que manejan el funcionamiento del datapath.
- *Memoria*: donde se guardan las instrucciones del programa y los datos utilizados por este..

Para la implementación del datapath se crearon los siguientes circuitos, donde se puede ver su conexión en Fig 1:

- Contador de programa: Un registro de 32 bits que guarda la dirección de la instrucción que se está ejecutando.
- Sumador: Un circuito combinacional que calcula la suma de dos operandos de entrada.

- Unidad de generación de datos: Un circuito combinacional que genera el campo de datos inmediato, de 12 o 20 bits, para las instrucciones aritméticas, lógicas, de carga, almacenamiento y saltos. Reorganiza la información del campo inmediato según el código de operación.
- Banco de registros: 32 registros de 32 bits de propósito general (excepto el registro x0, que se mantiene constante con el valor 0).
- Unidad Aritmético-Lógica (ALU): Un circuito digital que calcula operaciones aritméticas (suma y resta), lógicas (AND, OR) y comparaciones para saltos condicionales (set less than, SLT).

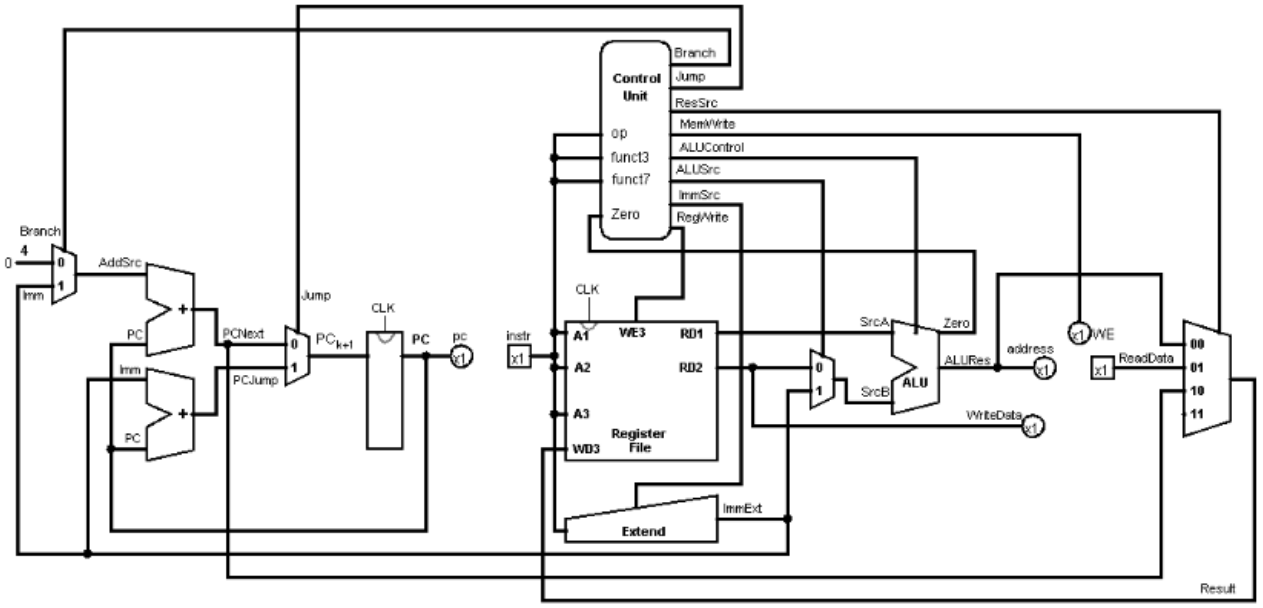

Fig. 1: camino de datos RV32i sin registros CSR

El mapa de memoria del RISC-V abarca un bus de direcciones de 32 bits, lo que permite un límite de 4 GB. Las direcciones de palabra son múltiplos de 4 y varían de 0 a 0xFFFFFFFC. El mapa de memoria divide el espacio de direcciones en cinco segmentos:

- Texto: Almacena el código del programa.
- Datos globales: Almacena datos estáticos.
- Segmentos de datos dinámicos: Almacena datos dinámicos.
- Controladores de excepciones: Almacena rutinas de manejo de excepciones.
- Sistema operativo (SO): Incluye memoria dedicada a la entrada/salida (E/S).

RISC-V no define un mapa de memoria específico, por lo que el usuario puede definir dónde se colocan estos segmentos.

En nuestro modelo simplificado, usamos una arquitectura de Harvard con un bus de direcciones de 16 bits, Esto implica que en el circuito, tengamos duplicados los buses de direcciones y de datos, pero nos brinda una mayor simplicidad en nuestra implementación.

Los circuitos de Memoria Implementados son (Fig 2):

- Memoria de programa: Implementa una memoria ROM con una capacidad de 512 bytes donde se aloja el programa que se está ejecutando.
- Memoria de datos: Implementa una memoria RAM con una capacidad de almacenamiento de 128 bytes donde el sistema aloja los datos utilizados por el programa (datos dinámicos y estáticos, pila).

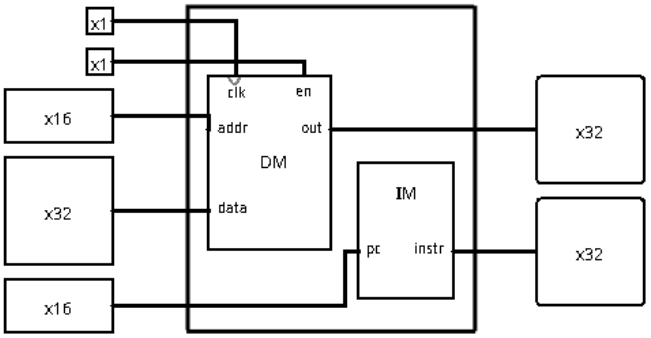

Fig. 2: módulo de memoria

La unidad de control es la circuitería que controla el flujo de datos a través del procesador, generando las señales necesarias para operar el datapath. Funciona como una máquina de estados finitos cuyas tareas son leer, decodificar, ejecutar la instrucción y almacenar los resultados. Para ello, se implementaron dos decodificadores:

- Decodificador principal: Decodifica el tipo de instrucción para generar el camino de datos (Tabla 1).
- Decodificador de ALU: Decodifica el tipo de operación para generar el control de la ALU (Tabla 2).

TABLA 1
tabla de verdad del decodificador principal

op	branch	jump	inm_s	mem_w	alu_s	reg_w	ALUOp
LW (3)	0	2'b01	2'b01	0	1	1	2'b00
SW (35)	0	2'b01	2'bxx	1	1	0	2'b00
R-type (51)	0	2'b01	2'b00	0	0	1	2'b10
B-type (99)	1	2'b01	2'bxx	0	0	0	2'b01
I-Type (19)	0	2'b01	2'b00	0	1	1	2'b10
jal (111)	0	2'b10	2'b10	0	x	1	2'bxx
desconocido	x	2'b11	2'bxx	x	x	x	2'bxx

TABLA 2
tabla de verdad del decodificador de operaciones aritmético-lógicas

ALUOp	F3	OP, F7	Instruction	alu_control
2'b00	x	x	**lw,sw**	000 (add)
2'b01	x	x	**beq**	001 (sub)
2'b10	3'b000	00, 01, 10	**add**	000 (add)
	3'b000	11	**sub**	001 (sub)
	3'b100	x	**slt**	101 (slt)
	3'b110	x	**or**	011 (or)
	3'b111	x	**and**	010 (and)

En la siguiente imagen podemos ver la concesión final entre los módulos

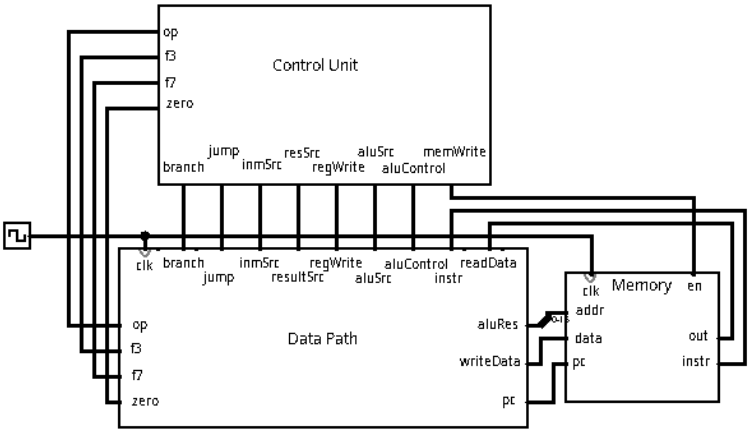

Fig. 3: conexiones entre los distintos módulos que conforman el procesador de ejemplo

III. Implementación del Manejador de Excepciones e Interrupciones

Una excepción es una llamada de función no programada provocada por un evento en el hardware o el software. Las excepciones de hardware, como las generadas por un dispositivo de entrada/salida (E/S), se denominan interrupciones. Por otro lado, las excepciones de software, como una instrucción indefinida, se denominan trampas.

Cuando ocurre una excepción, el control se transfiere a un programa encargado de resolver el problema (manejador de excepciones). Después de atender la excepción, el control se devuelve al programa que se estaba ejecutando, continuando como si nada hubiera ocurrido.

El controlador de excepciones utiliza registros de propósito especial, llamados registros de control y estado (CSR), para manejar una excepción. Los registros fueron implementados en el circuito *csr_reg*:

- *mstatus*: Habilita o deshabilita el manejo de excepciones mediante su bit menos significativo.
- *mtvec*: Contiene la dirección del código para el controlador de excepciones.
- *mepc*: Almacena el PC de la instrucción que generó la excepción.
- *mcause*: Registra la causa de la excepción.
- *mip*: Indica el estado de las interrupciones pendientes.
- *frm*: Sirve como bandera para la instrucción uret.

Cuando ocurre una excepción, el procesador registra la causa en *mcause*, almacena el PC en mepc, y salta al controlador de excepciones en la dirección configurada en *mtvec*. Después de manejar la excepción, el procesador retorna al programa principal ejecutando la instrucción *mret*.

Cabe aclarar que el manejador de excepciones debe usar los registros de propósito general (x1−x31) para manejar excepciones, por lo que existe el registro especial *mscratch* que contiene una dirección de memoria donde se almacena y restaura estos registros. Al ser un modelo simplificado y tener un modelo de memoria básica, se optó por no agregar este registro y usar una dirección de memoria común sin segmentar la unidad de memoria RAM.

RISC-V define tres instrucciones para leer, escribir o ambas operaciones en un registro CSR: *csrr*, *csrw* y *csrrw*. En la solución propuesta de diseño, se implementaron las instrucciones *csrrw* (lectura y escritura CSR) y *csrrwi* (lectura y escritura CSR usando un campo inmediato), a través de un decodificador de señales CSR en la unidad de control, podemos ver las señales que genera en la TABLA 3.

TABLA 3
TABLA DE VERDAD DEL DECODIFICADOR DE SENIALES PARA EL MODULO CSR

instruction	f3	csr_w	csr_data_s	data_read_sel	jump_mask
mret	3'b000	0	x	0	2'b11
csrrw	3'b001	1	0	1	2'b01
csrrwi	3'b101	1	1	1	2'b01
	otro caso	0	0	0	2'b11

Estas señales son usadas para habilitar o deshabilitar la escritura de los registros CSR (csr_w), como así también seleccionar la fuente la información entrante (*csr_data_s*) que puede ser leída desde un registro de propósito general o estar presente en un campo inmediato dentro de la instrucción, una seniat auxiliar (*data_read_sel*) que determina la fuente de que es guardada en el registro de propósito general. Por último, tenemos una máscara para la seniat *jump* generada en el decodificador principal de la unidad de control, para así gestionar el retorno a la rutina principal con la instruccion mret.

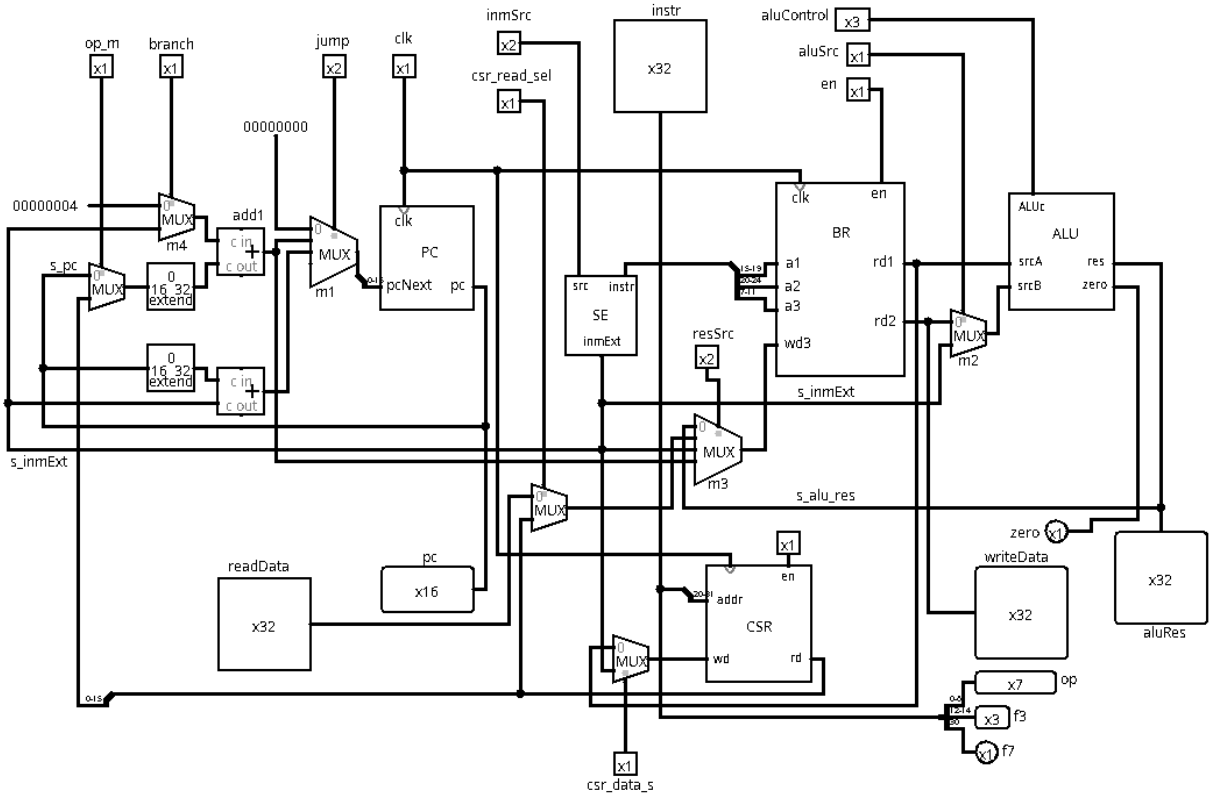

Fig. 4: Datapath que implementa modulo de registros CSR

IV. Manejo de Interrupciones

Se implementó un circuito (*verificadorExcept*) encargado de monitorear las señales internas del datapath y la unidad de control para detectar excepciones, interrupciones o trampas. Este circuito funciona siempre que el bit menos significativo del registro especial de estado *mstatus* esté activo y en caso de detectar algún evento, genera una señal de excepción que altera el flujo del datapath, modifica el valor del registro *mstatus* para evitar que se sigan atendiendo excepciones anidadas y por último, registra la causa y la dirección de la instrucción en ejecución en los registros CSR.

Se tomó la decisión de modificar el *mstatus* para desactivar la atención a excepciones y/o interrupciones, para asi evitar el anidamiento ya que esto implica una lógica más compleja y tener la necesidad de gestionar prioridades atención.

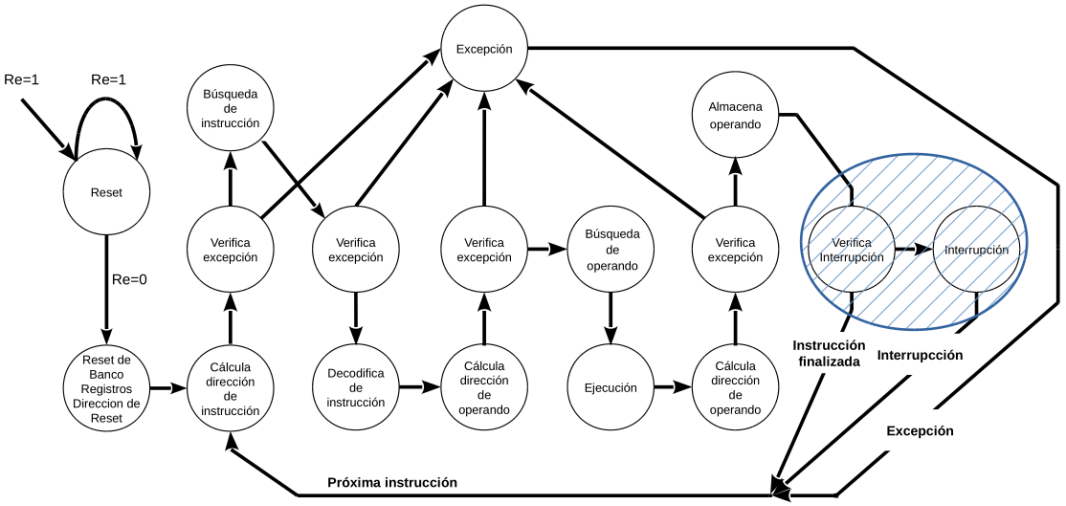

Fig. 5: Diagrama de estados del camino de datos para verificar excepciones e interrupciones

Como se muestra en Fig 5, este proceso de validación se realiza a lo largo de todo el pipeline de una instrucción, y por último se valida la atención a interrupciones externas.

Para atender interrupciones generadas por un pulsador, se modificaron los pines de entrada y salida de la unidad de memoria. Se definió un área de memoria dedicada a la entrada y salida (E/S), donde se conectan los periféricos. Se agregó una conexión directa a una dirección de memoria específica que almacena el estado del pulsador (activo o no), junto con un pin de salida que genera una señal IRQ al detectar un cambio de estado. Esta señal es capturada por el circuito verificador. Además, se implementaron dos pines de salida conectados directamente a memoria, los cuales se utilizan para controlar dispositivos externos, como LEDs, proporcionando una forma visual de monitorear el estado del sistema.

Este validador de excepciones, se limitó a identificar y atender un subconjunto de posibles causas de excepción, en la Tabla 4, se detallan las causas de excepciones/interrupciones que son registradas en *mcause*.

TABLA 4
VALORES MCAUSE CONTEMPLADOS EN EL DISEÑO PROPUESTO

Interrupción Bit 31 de mcause	Código de Excepción Bit 30:0	Descripción
1	11	Machine external interrupt
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	6	Store address misaligned
0	11	Environment call from M-Mode

V. Pruebas y Validación del Sistema

El sistema completo se validó configurando los registros CSR correspondientes para la atención de excepciones, interrupciones y traps. Posteriormente, mediante la ejecución de un programa contador en lazo cerrado que cada 10 vueltas del loop ejecutaba un ebreak y reinicia el contador, se probaron los dos escenarios principales:

- *Excepciones*: Generadas mediante un breakpoint en el código que se ejecutaba, en el cual se detenía por 6 ciclos de reloj adicionales la ejecución del programa principal al detectar una excepción de este tipo.

- *Interrupciones*: Generadas mediante un pulsador externo, se esperaba a terminar la ejecución de la instrucción actual y se modifica el flujo normal, atendiendo esta interrupción generando una detención de 3 ciclos de reloj adicionales el programa principal.

Los resultados (Fig. 6) demostraron que el procesador responde de manera correcta a estos eventos, pausando la ejecución y tomándola después de un intervalo de tiempo predefinido. También se pudo comprobar el correcto funcionamiento del registro *mstatus* que impide la ejecución anidadas de excepciones.

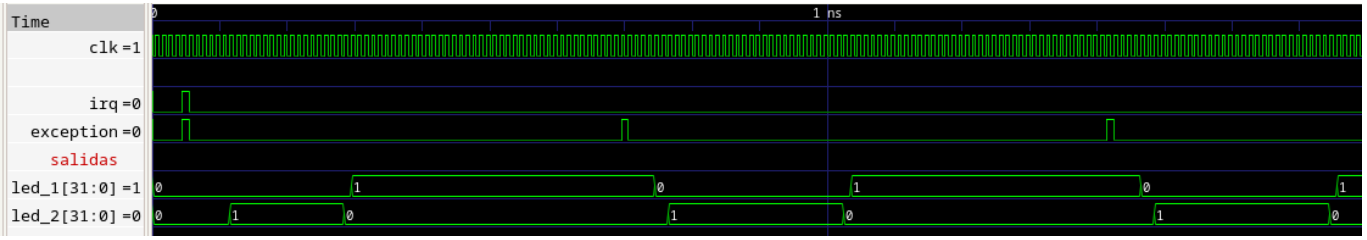

Fig. 6: Señales generadas por la prueba en testbech, muestra el estado de los puertos conectados a los LEDs

VI. Conclusión

El desarrollo de este procesador RISC-V con gestión de excepciones e interrupciones demostró ser un éxito, cumpliendo con los objetivos planteados. La implementación de los módulos CSR_regs y verificadorExcept permitió al procesador manejar eventos inesperados de manera eficiente, tanto aquellos generados por el software (como ebreak) como por el hardware (como la pulsación de un botón).

El programa contador en lazo cerrado utilizado para las pruebas demostró la robustez del sistema. Al alcanzar un punto de control específico, el procesador generaba una excepción mediante ebreak, pausando la ejecución y transfiriendo el control a la rutina manejadora de excepciones. Esta rutina guardaba el estado actual, realizaba operaciones específicas como los valores guardados en direcciones de memoria que sirven puertos de salida de información (a fin de por ejemplo activar y desactivar LEDs para indicar el estado del sistema) y retornaba la ejecución del programa principal después de un intervalo de tiempo predefinido.

De manera similar, la interrupción generada por el pulsador externo fue manejada correctamente, demostrando la capacidad del procesador para responder a eventos de hardware en tiempo real.

En resumen, este proyecto no solo cumplió con los requisitos técnicos, sino que también proporcionó una base sólida para futuras extensiones, como la implementación de más modos de operación o la integración de periféricos adicionales. La combinación de un diseño modular, una implementación eficiente y pruebas exhaustivas aseguró el éxito del procesador RISC-V desarrollado.