Trabajo Práctico Integrador 2

Bartezaghi Catriel y Katzenelson Germán
Facultad de Ingeniería y Ciencias Hídricas, catriel.b@outlook.com

***Resumen*—El presente informe expone la resolución del trabajo práctico integrador 2 de la materia, el cual consiste en implementar y codificar en lenguaje VHDL, un sistema que incluya el procesador monociclo RISC-V RV32I incluyendo el manejo de break points.**

***Palabras clave*—RISCV, RV32I, Procesador, Monociclo, VHDL**

I. Introducción

El procesador que vamos a desarrollar, es una ISA base. Y entre sus principales características es que cuenta con 32 registros de 32 bits y debe ser capaz de ejecutar las siguientes instrucciones: Tipo R (add, sub, and, or, xor, sll), I (addi, andi, ori, xori, slli, lw), S (sw), B (beq) y J (jal).

II. Implementación

Para orientar la implementación, utilizamos un enfoque top-down, por lo que partimos por definir los dos módulos principales, estos son el procesador y la memoria ram (fig. 1).

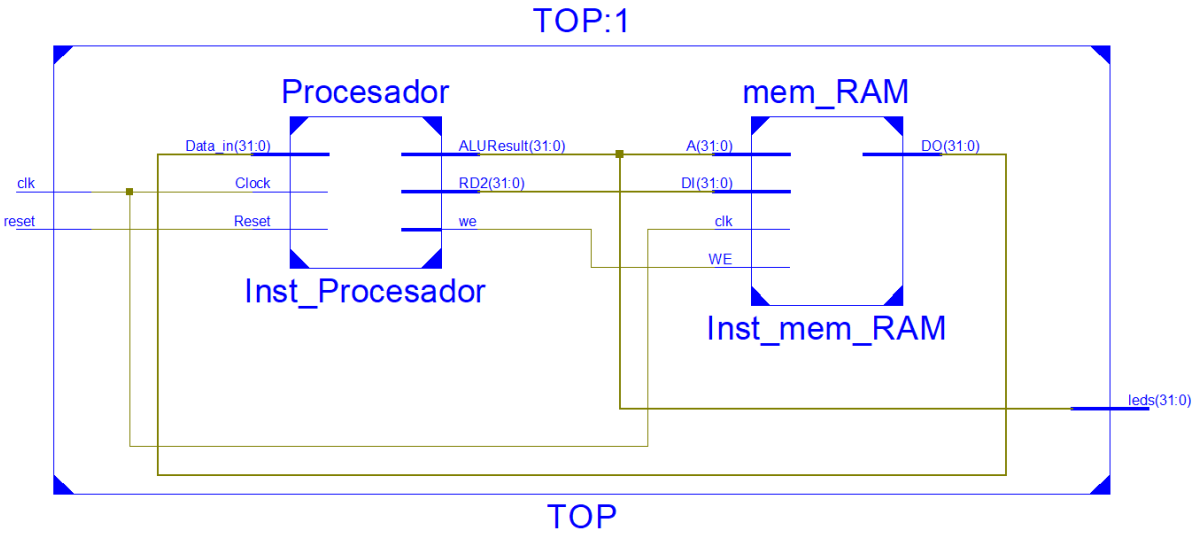

Fig. 1: Diagrama TOP

A. Procesador

El procesador está compuesto por dos módulos principales (fig. 2). Por un lado tenemos el camino de datos donde nos encontramos con la mayoría de los componentes del procesador, y por el otro la unidad de control, la cual se encarga de enviar las señales a los componentes del procesador de acuerdo a la instrucción que se quiere ejecutar.

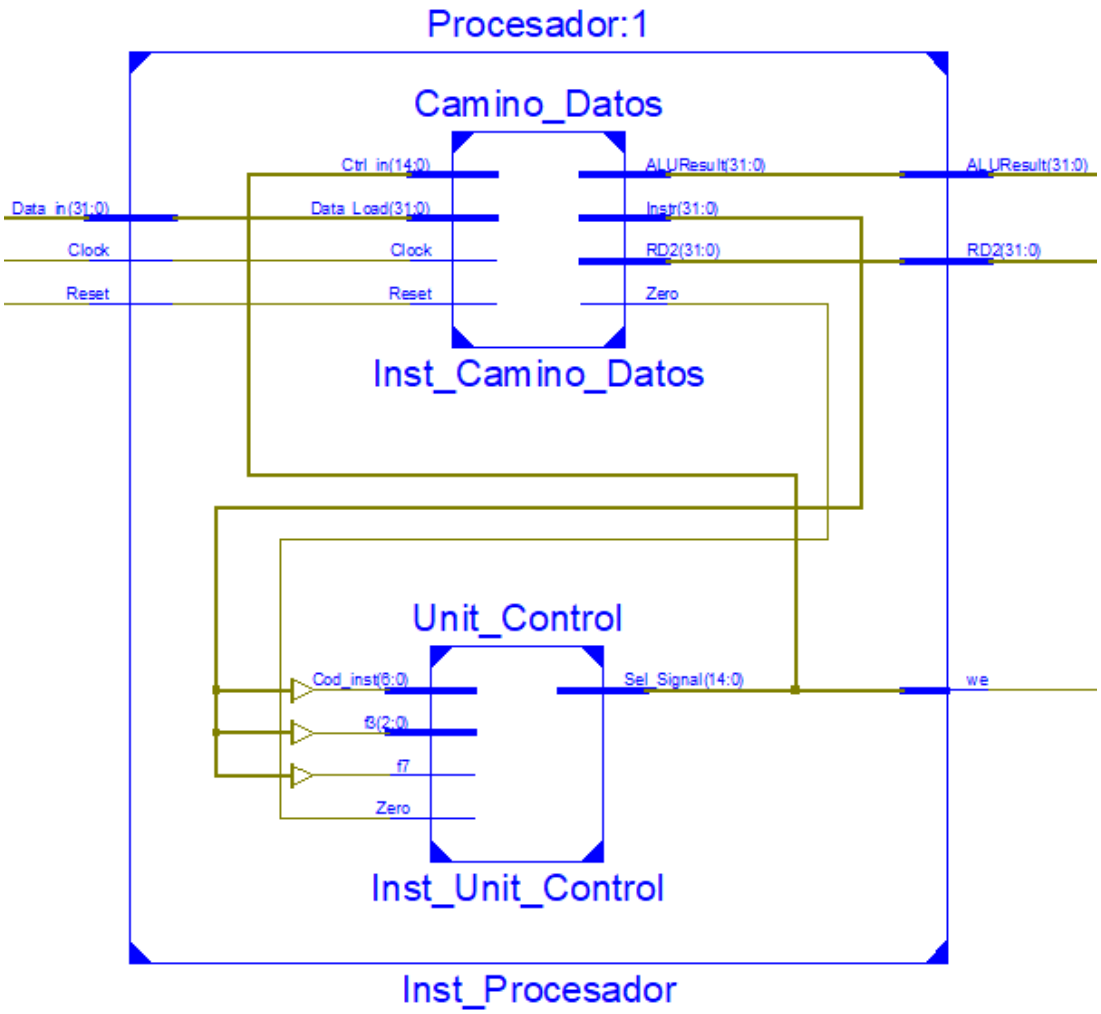

Fig. 2: Procesador

1. Camino de datos:

Este módulo comprende a la mayoría de los componentes del procesador (fig. 3). Para identificarlos y determinar cómo interconectarlos, se fueron evaluando los distintos tipos de instrucciones que se desean ejecutar. Se listan a continuación los componentes:

PC: Es el contador de programa que indica la posición del procesador en su secuencia de instrucciones. Actualiza la dirección de la instrucción en cada flanco de reloj. También tiene una entrada de reset para setear el contador en 0.

Instruction Memory (ROM): Contiene el programa, es decir todas las instrucciones a ejecutar. Tiene por entrada la dirección accedida, y por salida la instrucción a ejecutar. En nuestra implementación tiene cargada una secuencia de instrucciones a modo de prueba

Register File: Cuenta con varios registros, los cuales se utilizan de acuerdo a la instrucción ejecutada y a los valores que recibe, es el encargado de decodificar la instrucción.

ALU: Calcula las operaciones aritméticas, lógicas y las comparaciones para los saltos. Tiene por entrada dos operandos y el código que identifica la operación a realizar. Sus salidas son el resultado de la operación y una bandera de condición para cuando el resultado es 0.

Sum1: Suma 4 unidades al PC actual para avanzar con el programa.

Sum2: Recibe el valor de PC actual y un target, y los suma para posicionarse en un punto determinado del programa.

Extend: Reorganiza la información del campo inmediato de acuerdo al código de operación de la instrucción.

Multiplexores: Se utilizan para evitar cortocircuitos durante la ejecución del programa.

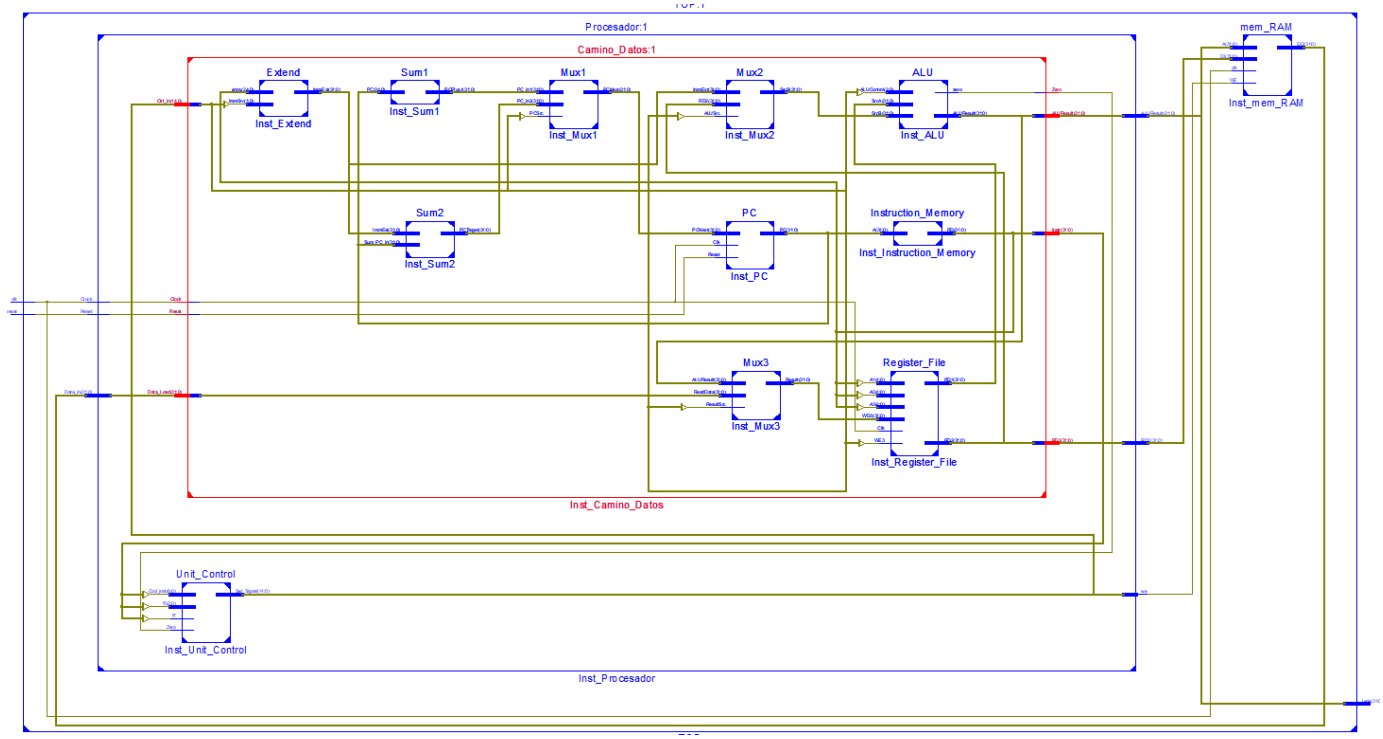

Fig. 3: Camino de datos sin la implementación de break points

2. Unidad de Control:

Se encarga de leer, decodificar, ejecutar las instrucciones y almacenar los resultados. Cuenta con un decodificador principal, el cual a partir del código de operación que recibe, envía las señales correspondientes a cada componente del sistema. Se puede visualizar mediante una tabla de verdad (tabla I).

Instrucción	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
Lw	1	00	1	0	1	0	00
Sw	0	01	1	1	0	0	00
R	1	00	0	0	0	0	10
Beq	0	10	0	0	0	1	01

Tabla I: Unidad de Control

Por otro lado, tenemos el decodificador de la ALU. Este recibe por parte del decodificador principal un código de operación, de acuerdo a la instrucción que se ejecuta y determina qué operación debe realizar la ALU (Tabla II).

ALUOp	Op	f3	f7	Instrucción	ALU
00	X	X	X	Lw, Sw	010 (add)
01	X	X	X	Beq	110 (subtract)
10	X	000	0	Add	010 (add)
10	1	000	1	Sub	110 (subtract)
10	X	010	0	Slt	111 (set less than)

10	X	110	0	Or	001 (or)
10	X	111	0	And	000 (and)

Tabla II: Decodificador de ALU

Finalmente tiene un operador lógico **AND** el cual tiene como entrada la salida del Main decoder **Branch** y la entrada de la unidad de control **Zero**, el resultado es una señal utilizada por el multiplexor 1 para realizar operaciones de salto.

Las salidas de la unidad de control son agrupadas en una sola señal **Sel_Signal** de 14 bits la cual es distribuida dentro del Datapath según su uso. (Tabla III).

Bit(s) de Sel_Signal	Señal representada
14	PCSrc
13	ResultSrc
12	MemWrite
11 – 9	ALUControl
8	ALUSrc
7 – 6	ImmSrc
5	RegWrite
4 – 0	00000*

Tabla III: Sel_Signal

*Donde nos reservamos los últimos 5 bits por alguna otra operación que necesite ser implementada en un futuro.

B. Memoria Ram

Se encuentra fuera del procesador. Aloja los datos utilizados por el programa. Tiene por entrada el resultado de la ALU y una habilitación de escritura que indica si debe guardar el dato, o enviarlo al banco de registros.

III. Desarrollo de Módulos para manejo de Excepciones

Para el manejo de break points en nuestro programa lo que hicimos fue traducir la instrucción **nop** del lenguaje Ensamblador a hexadecimal para saber cual es la representación que aparecerá en nuestro programa.

Teniendo esto en cuenta se agregaron 2 módulos, uno para la detección de una instrucción **nop** y otro para generar el retardo.

A. **Detección**:

Se incorporó un módulo a continuación de la memoria ROM, para detectar antes de distribuir a los demás componentes una instrucción que, según Wikipedia, "*no hace nada*". Al notar que la señal es un x"00000013", emite una señal **Sen_ret** seteada en 1 la cual llega a nuestro siguiente nuevo componente, a su vez, lanza una instrucción x"00000000" "simulando" ser la memoria ROM, la cual mediante unos test bench comprobamos que no afectan el desarrollo del programa.

B. **Retardador**:

El segundo módulo, se sitúa entre la salida de PC y los 2 sumadores. Tiene 3 entradas, un Clock, la salida de PC y la señal Sen_ret que sale de Verif_Instr. Mientras esta última señal permanezca en 0, la salida del retardador será la de PC continuando el flujo normal de datos, en cambio al setearse en 1, mediante un ciclo que opera con el reloj del RISC y teniendo en cuenta que cada flanco ascendente se da cada 10ns, se llegan a 300000000 iteraciones con las cuales la ejecución se pausa por 3 segundos completos, una vez completados se lanza en la señal de salida el vector que originalmente ingreso desde PC.

El uso del retardados trae varios beneficios como que en caso de que una excepción requiera correr un cierto código, se puede configurar para que el número del contador que sale por el módulo sea el de la solución al problema.

Finalmente, nuestro diagrama del procesador RISC-V RV32I con los módulos de execepciones queda compuesto de la siguiente forma. (Fig 4)

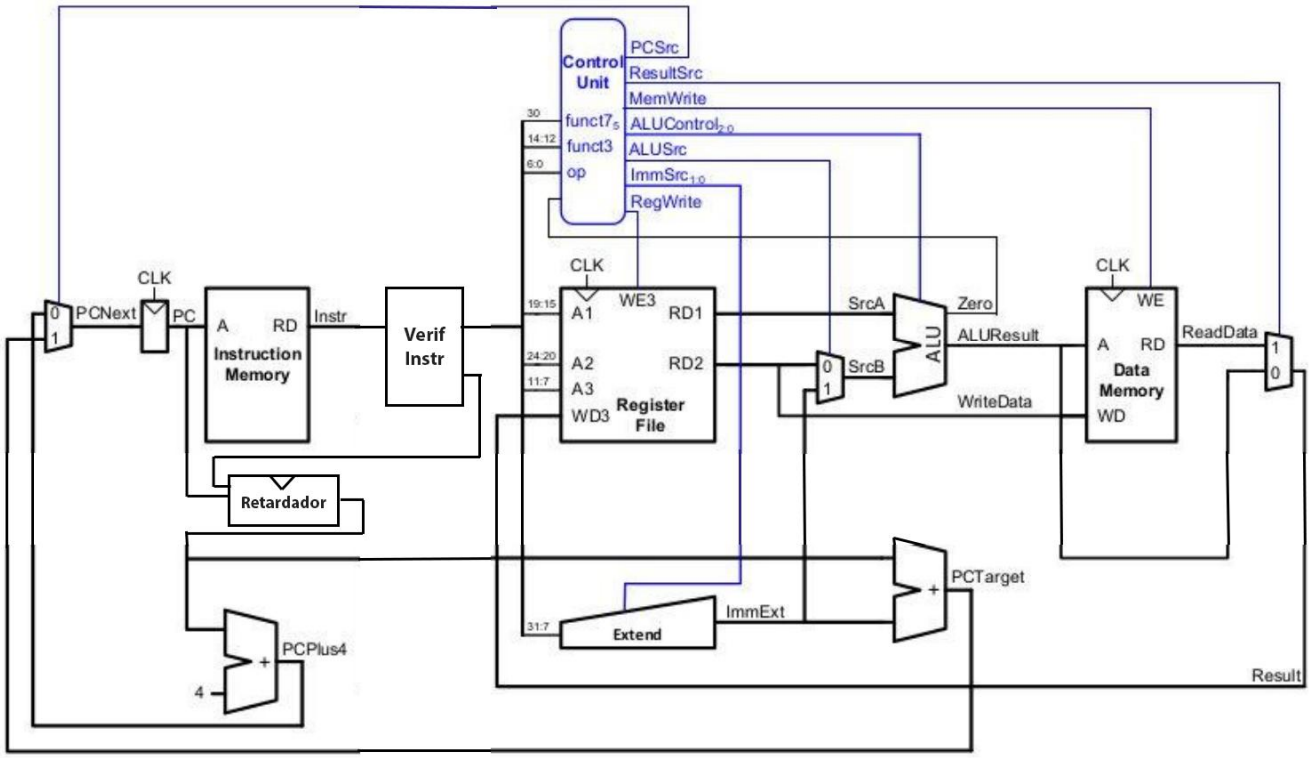

Fig. 4: Diagrama RV32I

IV. CONCLUSIÓN

El trabajo fue muy enriquecedor y gracias a las herramientas aprendidas en la materia pudimos crear ni mas ni menos que nuestro propio procesador monociclo. El agregado del retardador para el manejo de excepciones puede ser expandido para un uso más específico como correr solucionadores de ciertas partes de código o como en este caso, retardadores.