Organización de las Computadoras Trabajo Práctico Integrador

Bargas, Santiago Darío. Bode Ignacio. Cassiet, Marcelo Tomas.
Universidad Nacional del Litoral, Facultad de Ingeniería y Ciencias Hídricas

***Resumen*— En el siguiente informe se presenta una explicación del proceso de desarrollo de un sistema integrado en Verilog y Assembly para realizar distintas operaciones como addi, beq, add.. etc.**

En la primera sección, que es la introducción, se explica la consigna presentada por la cátedra y se realiza una breve descripción del sistema integrado propuesto. En las siguientes secciones, se explican las decisiones que se tomaron para definir las bases del diseño de los componentes del sistema.

En la última sección se presentan las conclusiones obtenidas tras haber finalizado el trabajo.

I. Introducción

A. Consigna

La consigna del Trabajo Práctico Integrador consiste en la verificación y prueba de funcionamiento de un procesador diseñado según la arquitectura rv32i de RISC-V. Habiendo diseñado el procesador rv32i, el siguiente paso es realizar una prueba exhaustiva de sus instrucciones para asegurar su correcto desempeño. Para ello, se empleará un código de ejemplo que, tras algunas modificaciones específicas adaptadas a nuestra implementación, permitirá verificar el funcionamiento en un simulador.

Este código incluirá instrucciones fundamentales de RISC-V que evaluarán las operaciones aritméticas, lógicas y de control de flujo, entre otras.

El objetivo final es cargar este código en la memoria de instrucciones y simular el comportamiento del rv32i.

II. Diseño e implementación del procesador

El procesador propuesto por la cátedra es el RV32I, que fue construido durante el cursado y consta de dos módulos: el datapath (camino de datos) y la unidad de control. Además, por decisiones de diseño, hemos agregado un módulo adicional llamado Memory. Este módulo se encarga específicamente de gestionar el almacenamiento y la recuperación de datos. A continuación explicamos los módulos realizados.

A. El datapath

El módulo ***datapath*** se encarga de gestionar el camino donde fluyen los datos a medida que se ejecutan las instrucciones, realizando las operaciones aritméticas y/o lógicas necesarias para ejecutar las instrucciones del programa.

El *datapath* cuenta con los siguientes módulos:

- **Contador de programa (PC)**: registro de 32bits que actúa como el contador de programa. La salida (`pc`) se actualiza con el valor de entrada (`pcNext`) en cada flanco positivo del reloj (`clk`).
- **Banco de registros (BR)**: banco de 32 registros de 32 bits. Las entradas (`a1, a2, a3, wd3`) determinan las operaciones de lectura y escritura, y `we` indica si se debe escribir. Las salidas (`rd1, rd2`) contienen los datos leídos.
- **Extensión de signo (SE)**: se encarga de extender el bit de signo de la entrada inmediata (`imm`) a 32 bits.
- **Unidad Aritmético Lógica (ALU)**: realiza las operaciones aritméticas y lógicas necesarias para implementar las instrucciones mencionadas anteriormente. La salida `res` contiene el resultado de la operación.
- **Unidad Aritmético Lógica Suma (ADDER)**: realiza la suma de dos operandos (`op1` y `op2`), utilizada para operaciones específicas en el camino de datos
- **Multiplexores 2x1 y 3x1**: multiplexores de 2 a 1 y 3 a 1 de 32 bits. Este tipo de módulo selecciona una de las dos o tres entradas basándose en la señal de selección (`sel`).

B. La unidad de control

El módulo **unidad de control** se encarga de generar señales de control de los distintos módulos del flujo de datos para manejarlo. Cuenta con los siguientes sub-módulos:

- **Decodificador de operación principal (mainDeco)**: toma el campo de operación (`op`) de una instrucción en el formato RV32I

y genera señales de control para dirigir el camino de datos del procesador. Estas señales se definen de acuerdo a la siguiente tabla:

op	Ins	RW	IS	As	MW	RS	B	Ao	J
3	lw	1	00	1	0	01	0	00	0
19	addi	1	00	1	0	00	0	10	0
35	sw	0	01	1	1	xx	0	00	0
51	R-t	1	xx	0	0	00	0	10	0
99	beq	0	10	0	0	xx	1	01	0
111	jal	1	11	x	0	10	0	xx	1

Las columnas representan los valores de: campo op de la instrucción, nombre de la instrucción, RegWrite, ImmSrc, ALUsrc, MemWrite, ResultSrc, Branch, ALUop y Jump.

- **Decodificador de ALU**: toma señales de control específicas y genera las señales de control necesarias para configurar la ALU de acuerdo con la instrucción que se está ejecutando. Estas señales se definen de acuerdo a la siguiente tabla:

Instrucción	f3	op_5	$f7_5$	$f7_0$	$ALUctrl_{2:0}$
lw	xxx	x	x	x	000
addi	xxx	x	x	x	000
sw	xxx	x	x	x	000
add	000	1	0	0	000
mul	000	1	0	1	100
sub	000	1	1	0	001
slt	010	x	x	x	101
div	100	x	x	x	110
or	110	1	0	0	011
rem	110	1	0	1	111
and	111	x	x	x	010
beq	xxx	x	x	x	001
jal	xxx	x	x	x	000

- **Generador de señal de control (pcSrc)** : se utilizan las señales zero y Branch para decidir si la próxima dirección del contador de programa se toma del resultado de una operación de salto condicional.

C. Unidad de memoria (memory)

El **módulo de memoria** se encarga de la lectura de instrucciones y la carga y almacenamiento de datos. Se encarga de gestionar la memoria utilizada por el procesador rv32i, separando las instrucciones y los datos en dos unidades de almacenamiento: **Memoria de Instrucciones** (IM) y **la Memoria de Datos** (DM)

Está compuesto por los siguientes sub-módulos:

- **Memoria de instrucciones (IM):** memoria de sólo lectura de 32 registro de 32 bits. Almacena las instrucciones que el procesador ejecutará secuencialmente. Su entrada (addresIM) es una dirección en la memoria de instrucciones que permite acceder a una instrucción específica. Su salida (inst) es una instrucción obtenida de la memoria, que se envía a la unidad de control para su decodificación y ejecución. Aquí se cargara el programa antes de iniciar la simulación, para ello se realizo un código en RARS, configurando el Compact Data at address en 0 para poder usar las instrucciones lw y sw desde un puntero a la memoria de datos en la posición 0x00000000 como en nuestra implementación. Este código se exporto en formato hexadecimal y se cargo manualmente en la memoria.
- **Memoria de datos (DM)**: almacena datos que el procesador necesita para la ejecución de instrucciones, especialmente las de carga (load) y almacenamiento (store). Entradas: (addresDM) dirección en la memoria de datos donde se realizará la lectura o escritura. (wd) datos que se escribirán en addresDM si la señal de escritura (we) está activa. (we) señal de habilitación de escritura Salidas: (rd) Datos leídos de la dirección especificada en addresDM, que serán usados en operaciones dentro del datapath

A continuación se presenta el esquema de los módulos separados, y la composición del modulo rv32i elegida para el diseño del procesador, los cuales fueron implementados para el trabajo

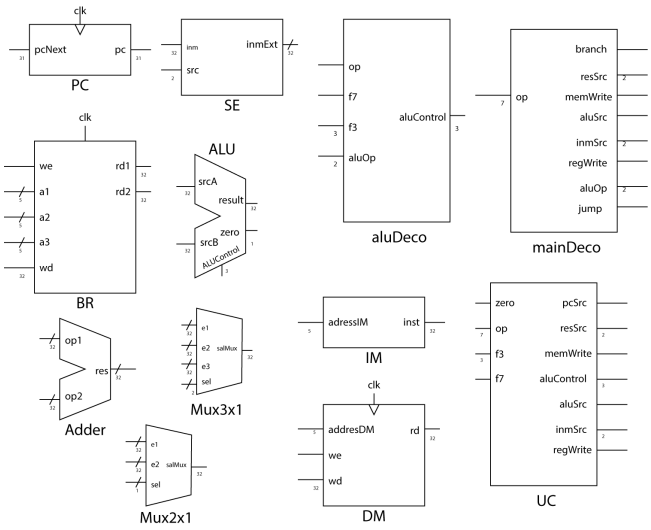

Fig. 1: Esquema de módulos separados

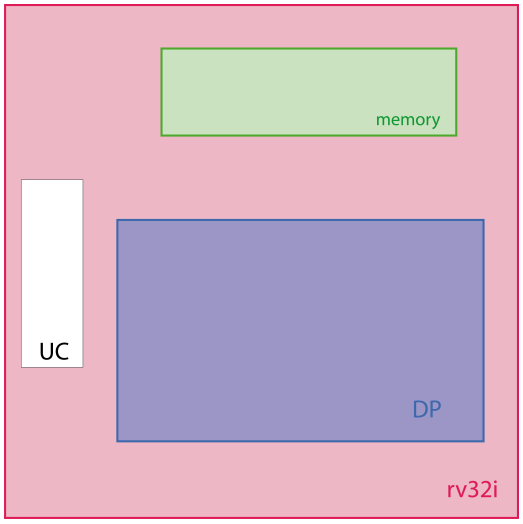

Fig. 2: Diseño rv32i sin conexiones

III. Test Bench (Banco de pruebas)

El Test Bench (banco de pruebas) es una herramienta fundamental en el desarrollo y verificación de sistemas de hardware, como el procesador rv32i en este caso. Su propósito es emular el entorno de funcionamiento del procesador en un entorno de simulación, permitiendo evaluar el comportamiento del diseño y verificar que todas las instrucciones y operaciones funcionen correctamente.

Al finalizar la prueba, los resultados observados en el Test Bench permiten confirmar si el procesador rv32i está listo para su implementación o si requiere ajustes adicionales. En conclusión, el Test Bench es una etapa crítica en el proceso de desarrollo de hardware, ya que asegura la confiabilidad y funcionalidad del procesador antes de su despliegue en un entorno físico o de producción.

A continuación vemos un ejemplo de la simulación para corroborar el correcto funcionamiento:

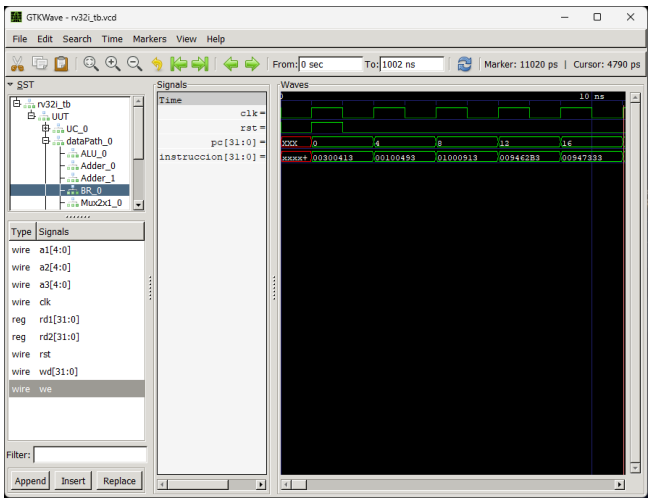

Fig. 3: Test Bench: clk, rst, pc, instrucción

En la Fig. 3 se puede observar el clk, el rst, el contador de programa que va incrementando de 4 en 4 y las instrucciones que se van ejecutando.

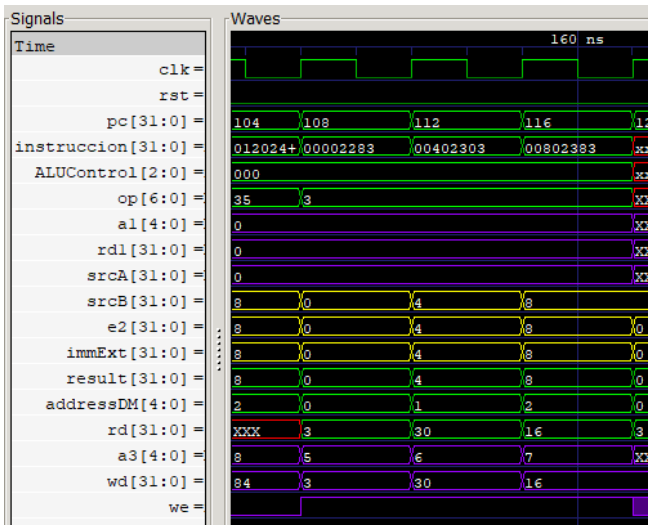

Fig. 4: Test bench: Final del programa

En la Fig. 4 se puede observar los resultados finales de los valores de los registros t0,t1 y t2 los cuales coinciden con los valores esperados. Por lo tanto podemos decir que el procesador esta funcionando correctamente

Por ultimo, en la Fig. 5 se presenta el esquema de la arquitectura completa con los módulos conectados

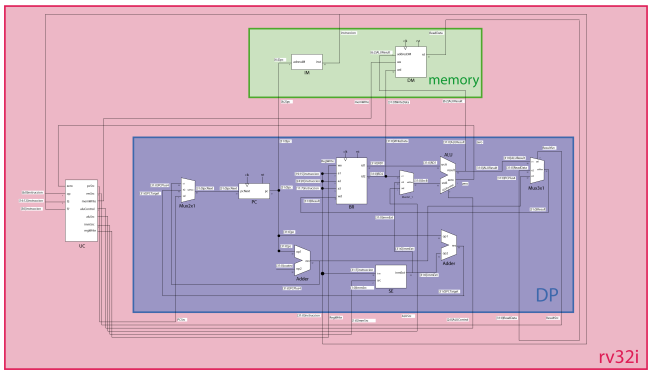

Fig. 5: Módulos Conectados

IV. Conclusiones

El desarrollo e implementación del procesador rv32i en Verilog ha permitido comprobar el correcto funcionamiento de un diseño basado en la arquitectura RISC-V de 32 bits. La implementación modular, compuesta por el datapath, la unidad de control, y el módulo adicional de memoria (Memory), ha demostrado ser efectiva para gestionar de manera ordenada tanto el flujo de instrucciones como los datos. Tras el proceso de diseño y simulación, se ha verificado que el

procesador es capaz de ejecutar correctamente las instrucciones clave de rv32i. Las operaciones aritméticas, lógicas, y de control de flujo se procesan como se espera, reflejando los resultados correctos en los registros y la memoria. Esto confirma la solidez de la estructura de nuestro procesador y la efectividad de las decisiones de diseño, como la incorporación del módulo de memoria para una mejor organización del almacenamiento. El funcionamiento actual indica que el diseño del procesador cumple con los requerimientos del conjunto de instrucciones rv32i, evidenciando una correcta integración de todos los módulos involucrados.