Implementación de una microarquitectura para el RISCV

Álvaro Méndez Ceciliano
e-mail:Mendez.ceciliano@gmail.com
Adriana Rojas Mesén
e-mail:adrianarm31@gmail.com
Cristofher Solís Jiménez
e-mail: cristofhersj@gmail.com
José David Soto Zuñiga
e-mail: josedavidsz@hotmail.com

ABSTRACT: *The present work consists of the design and implementation of a microarchitecture for the RISCV instruction set (ISA). Implemented using Verilog Hardware Description Language (with Xilinx's ISE platform). The objective is to familiarize with the development of microarchitectures and all that this entails (design, digital tools and concepts). It was not possible to obtain a circuit capable of complying with instructions with type R, I, S, B, J and U decoding but the modules that composed the architecture were designed, implemented and tested individually with success* .

Keywords: 1. Datapath, 2. Instruction set architecture (ISA), 3. Memory, 4. Microarchitecture, 5. Register.

I. INTRODUCCIÓN

El uso de dispositivos electrónicos ha crecido exponencialmente en las últimas décadas, por tanto la demanda de microprocesadores también ha aumentado y el uso de los mismos se ha diversificado. Respecto a la arquitectura empleada por estos componentes electrónicos, destacan dos grandes grupos: *complex instruction set computer* (CISC) y *reduced instruction set computer* (RISC). La diferencia entre estos radica respectivamente en que, mientras el primer grupo busca minimizar la cantidad de instrucciones por programa (aunque esto signifique mayor complejidad en las instrucciones), el segundo grupo busca lo contrario, utilizar instrucciones simples a costa de requerir más instrucciones para cumplir el objetivo.

Dentro de las arquitecturas más destacadas para CISC y RISC respectivamente están x86 y ARM. Sin embargo, el uso de las mismas requiere licencias que añaden costo final al producto. Nuevas tecnologías han planteado nuevos retos en la industria de semiconductores, siempre en busca de precios competitivos.

Una ISA de código abierto tipo RISC llamada RISCV, ha llamado la atención pues su uso es gratuito y se ha empezado a popularizar, en especial por su naturaleza gratuita.

En el presente trabajo se profundiza en el diseño e implementación de una microarquitectura basada en el ISA RISCV32I con el propósito de familiarizarse con dicho proceso y conceptos asociados a los microprocesadores.

Primeramente se ahondará en el diseño de las distintas etapas de los circuitos -módulos- que describen la microarquitectura implementada, tomando en cuenta la documentación oficial [1], para lograr soportar las decodificaciones R, I, S, B, J y U. Cabe destacar que al ser RISCV32I se contará solamente con 32 registros.

Seguidamente se detallará mediante un datapath, la justificación de algunas de las decisiones de diseño tomadas para concretar correctamente dicha ruta de datos. Asimismo se adjuntará al final del presente documento circuitos complementarios para un apropiado entendimiento de la implementación.

Es importante rescatar que una de las metas de este proyecto consistía en la lectura de seis códigos escritos en C. Mediante un toolchain se logró obtener un dump de cada C, y de esta forma se trasladó cada instrucción de interés al archivo de texto.

Se espera que este trabajo sea de ayuda para personas con interés en implementar microarquitecturas de esta naturaleza y/o de paso comprender conceptos o aspectos referentes a RISCV.

II. DISEÑO

B. Decodificación

En RISC-V la decodificación es distinta para cada tipo de instrucción, esto se dicta según la siguiente Figura:

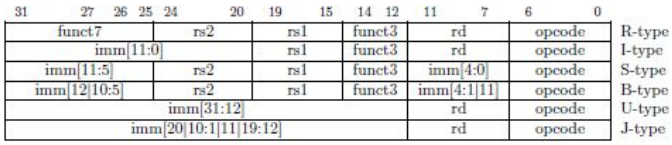


Figura 1. Tabla de decodificación para los distintos tipos de instrucciones en RISC-V

Para lograr cumplir con el código necesario se realizó un dump de las instrucciones requeridas en nuestra implementación, se definieron como necesarias las instrucciones denotadas en la tabla 1. Cada una se trató de la debida forma para lograr hacer la decodificación correcta de cada una.

La instrucción JALR posee la excepción de pertenecer al tipo de instrucciones J pero realiza su manejo de inmediatos de manera similar a las instrucciones tipo I

Tabla 1: Lista de instrucciones para el microprocesador diseñado.

Tipo	Instrucción
R	Add
	Sub
I	Addi
	XORi
	ANDi
	SLLi
	SRAi
	SLRi
	LBU
	LW
S	SW
	SB
B	Bne
J	Jalr (I)
	Jal
U	LUI

A. MÓDULOS

Manipulador de instrucciones: La microarquitectura debe ser capaz de soportar operaciones que involucren saltos (por ejemplo JAL o BNE) para ello se debe contemplar que existe un registro encargado de llevar el control de cuál instrucción se realizará: registro PC. Este registro será quien brinde una dirección de memoria para el banco de instrucciones. El flujo normal del programa (pasar a la siguiente instrucción) estará dado por PC+4, en caso de existir un salto, la dirección será PC+inmediato. El módulo contempla dichos escenarios (incluyendo el caso de la instrucción que realiza un salto en caso de presentarse una desigualdad: BNE), gracias al uso multiplexores cuyos selectores serán señales de control así como la consideración del resultado de comparación para el caso de BNE.

Banco de instrucciones: Corresponde a una memoria ROM capaz de cargar directamente las instrucciones guardadas en un archivo de texto. En dicho archivo cada instrucción se separa por un *enter*. Es importante destacar que cada instrucción consta de 32 bits los cuales pasarán a una sección de decodificación. Asimismo, la cantidad de de direcciones de memoria está dada por 2^11 (2048 posibles direcciones de memoria), esto con el fin de reducir carga en el procesamiento del módulo y reducir tiempos de espera en simulación.

Otro aspecto por destacar es el hecho de que las instrucciones base para probar la implementación fueron adquiridas del dump file de distintos archivos programados en C (desde un toolchain de RISCV32I instalado en un sistema operativo Ubuntu). Este elemento,

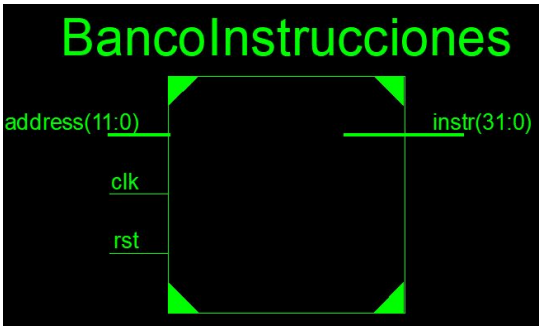


Figura 2. Banco de instrucciones.

Register File: Este módulo tiene la función de cargar y guardar datos en registros, en este caso específico, se cuenta con 32 registros, 6 entradas y 2 salidas, las entradas constan de:

D1: Entrada de 32 bits, consta de los datos que se desean guardar o tratar en algún registro.

R1: Es una entrada que dicta cuál es el número del primer registro a utilizar.

R2: Es una entrada que dicta cuál es el número del primer registro a utilizar.

WEn: Se trata de una señal que habilita la función de guardar datos en un registro.

W1: Esta señal dicta en cuál registro se van a guardar los datos.

clock: es el reloj del sistema, a partir de este es que funciona el módulo.

Out1, Out2: Son las 2 salidas, según la instrucción que se está aplicando se utiliza una, otra o ambas.

Por ejemplo, en el caso de Add R1, R3, R2; El register File guardaría el resultado en el registro R1 y le mandaría los datos de R3 y R2 a la ALU.

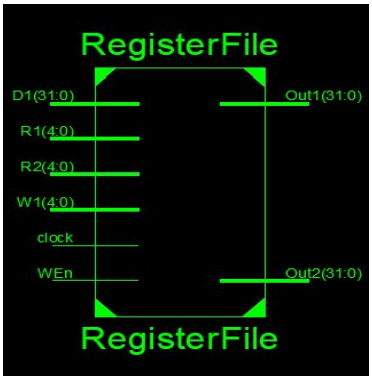


Figura 3: Esquemático del módulo de Register File

Extensor de Signo: La finalidad de este módulo es encargarse del manejo de inmediatos a la hora de la decodificación de instrucciones, cada tipo de instrucción maneja de forma distinta sus inmediatos, por lo que es un componente relativamente complejo debido a la variedad de su salida con respecto a las entradas que recibe, consta de 4 entradas y una única salida, las cuales se denominan:

COD: se trata del código que recibe por parte de la instrucción, se decodifica la parte de los inmediatos y se re acomoda para la salida, la lógica que utiliza es la que se puede ver en la figura 1.

Type: Es una entrada que dicta cuál es el tipo de instrucción que se está llevando a cabo.

JALR: como se mencionó anteriormente, Jalr es una instrucción que tiene una decodificación única, por lo que esta señal permite diferenciar cuando se está corriendo dicha instrucción en específico

clock: es el reloj del sistema, a partir de este funciona el módulo

En casos de instrucciones de tipo J, se ordenan los bits de la decodificación de la instrucción y acomoda los datos que corresponden al inmediato como se denota en la Figura 1.

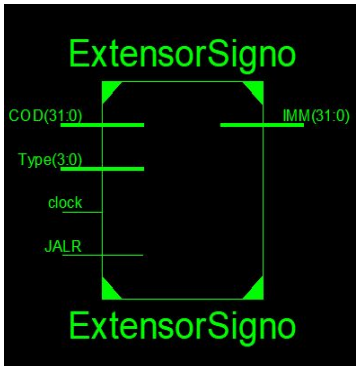


Figura 4: Esquemático del módulo del Extensor de Signo

Unidad de Control: Este módulo se encarga de manejar las distintas señales de control que se van a necesitar para el correcto funcionamiento del microprocesador, consta de tres entradas y nueve salidas, las cuales se denominan como:

Opcode, Funct3 y Funct7 : las tres señales de entrada se dan directamente de la decodificación de la instrucción a realizar, entonces se toman los segmentos de esa decodificación conocidos con los mismos nombres de las señales de entrada.

alu_sel: Controla el modo de operación de la ALU utiliza cuatro bits para seleccionar entre las siguientes operaciones

Tabla 2. Operaciones de la ALU y su código respectivo de Alu_sel

0000	Suma
0001	Resta
0010	AND
0011	XOR
0100	SRA
0101	SLL
0110	SRL
0111	BNE
1000	LUI

alu_src: Controla el feed de una de las entradas de la ALU, para poder operar entre inmediatos, registros o directamente el PC+4 actual

ext_sel: Le indica al extensor de signo el tipo de inmediato que debe procesar según la siguiente tabla

Tabla 3.Tipos de inmediato manejados por el extensor de signo si su código respectivo de ext_sel

0000	TIPO R
0001	TIPO I
0010	TIPO S
0011	TIPO B
0100	JAL
1100	JALR
0101	TIPO U

reg_sel: Selecciona el feed que recibe la entrada de información del register file entre la salida de la ALU, la salida del bloque de memoria y el pc+4

jal_sel: Indica que la instrucción actual es un salto de tipo branch para permitir que el avance de pc dependa de la salida de la ALU

j_sel: controla el feed del bloque de PC seleccionar la nueva dirección entre el resultado de la ALU o el sumador de PC

width: Indica el tamaño de la información para un STORE o LOAD entre un byte o una palabra entera

WS_en: Habilita escritura en la memoria

W_en: Habilita escritura en el Register File

Cabe destacar que la unidad de control es de suma relevancia para todo tipo de decodificación, pues es la encargada de seleccionar las señales de interés a la hora de querer realizar correctamente una instrucción. Por ejemplo, permitiría escoger entre la utilización de un inmediato en lugar de la información proveniente de un registro para una operación como ADDI o por el contrario, el uso de la información del registro en una operación ADD.

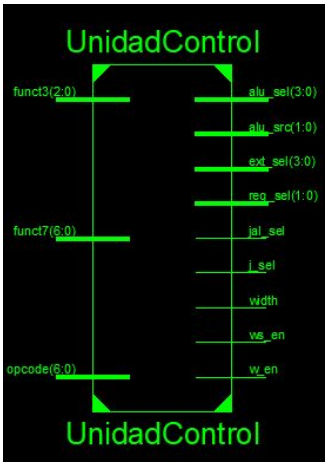


Figura 5: Esquemático del módulo de la Unidad de Control

Unidad aritmético lógica (ALU): Este módulo corresponde a la unidad encargada de realizar operaciones aritméticas y lógicas. Presenta dos entradas, cada una de 32 bits. Asimismo presenta un selector de 4 bits capaz de seleccionar cuál instrucción se va a realizar. Dicha señal de selección proviene de la unidad de control.

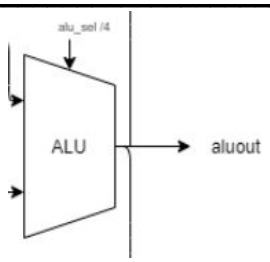


Figura 6: Esquemático del módulo de la ALU

Las operaciones así como su respectiva señal de selección se encuentran en la tabla xx. En este caso se hará referencia a una instrucción ADD, en dicho caso se tomarán ambas señales de entrada y se sumarán (alu_sel=0000). En la sección de anexos se podrá observar el datapath de esta instrucción.

Memoria:

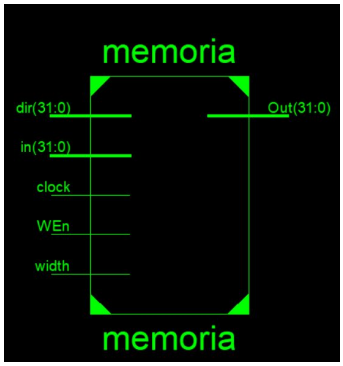


Figura 7: Esquemático del módulo de la Memoria

El bloque memoria ilustrado en la figura 7 posee como entradas las variables dirección(dir) que controla la dirección de memoria relevante para la operación actual, información (in) que acarrea la información para ser almacenada si la operación es de tipo STORE, write enable (WEn) que indica que la operación es de tipo STORE y permite la escritura, width qué indica el tamaño de información que se espera procesar y clock que permite llevar el orden de la escritura a través del reloj del sistema, posee únicamente una salida (Out) la cual refleja el valor almacenado en la dirección indicada, esta salida es únicamente útil si la operación actual es de tipo LOAD

Multiplexores para selección de señales: A continuación se van a presentar multiplexores encargados de seleccionar la señal correspondiente según el tipo de instrucción. Generalmente se colocan pues el mismo componente podría tener como entrada señales distintas según la operación. Dentro de los casos que se encuentran en la microarquitectura están

-Mux. para selección de entrada de ALU: la ALU puede realizar operaciones empleando la información contenida entre dos registros o bien, entre un registro y el concatenador de inmediatos. Para dicha selección se emplea un multiplexor.

-Mux para la selección de entrada de información por escribir en registro: previo a la señal en escritura se considera si guardar la información que viene de memoria o de la suma de PC+inmediato o directamente de la ALU.

GPIO: este módulo consiste internamente en un contador que se encargará de contar las ocasiones en que la señal de dirección que entra a la memoria de datos obtenga el valor de 0xABCD. Una vez el contador alcance el valor de dos, se tomará el valor de Out2 del banco de registros y este será mostrado en la salida del módulo. Cabe destacar que este consiste en la salida del sistema. Nótese que es susceptible a la señal del reloj (flanco positivo).

III. RESULTADOS

Se realizó un dump de las instrucciones necesarias para la ejecución de los seis archivos .C brindados para el proyecto con lo que se construyó un diseño teórico de una arquitectura capaz de soportar el conjunto de instrucciones representado en la tabla 1.

Se demostró mediante pruebas individuales a través de un testbench el funcionamiento de los módulos de memoria, recortador, banco de registros, extensor de signo, ALU y Unidad de Control.

No obstante, en el módulo de implementación del microprocesador como tal, se vio impedido por el "Warning: XST:1290", el cual desactiva los módulos en que se encuentra la vulnerabilidad que detecta, por lo que el módulo principal no posee la mayoría de los módulos probados con anterioridad debido a esta advertencia y su manera de funcionar, no se logró obtener una solución eficaz en el tiempo requerido, por lo que no se logró implementar la arquitectura.

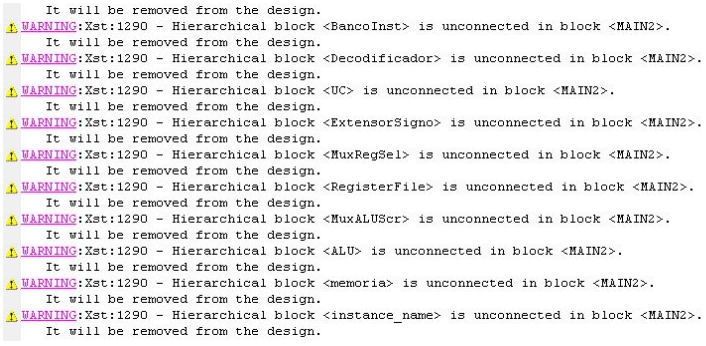


Figura 8: Warning: XST:1290

hardware Verilog. Sin embargo, se plantearon distintos módulos funcionales que componen la microarquitectura.

El banco de instrucciones logró hacer la lectura de archivos de texto obtenidos mediante el dump de los códigos en C. Es decir, para cada dirección que se le asignara, la salida correspondía efectivamente a la instrucción pertinente. Asimismo la unidad de control UC permitía desplegar las señales de control necesarias para el correcto manejo de las distintas decodificaciones.

V. CONCLUSIONES

No se pudo obtener una implementación funcional para la microarquitectura propuesta mediante el lenguaje de descripción de hardware Verilog. Sin embargo, se plantearon distintos módulos funcionales que en conjunto componen la microarquitectura.

La interacción de una implementación en verilog es compatible con archivos externos como .txt, como se demostró al cargar la información del banco de instrucciones.

Efectivamente se logró la familiarización con distintos conceptos y lógica de RISCV, así como del diseño en términos generales de una microarquitectura y del trabajo que esto conlleva en términos de diseño digital y herramientas de programación.

VI. RECOMENDACIONES

Antes de iniciar el diseño e implementación de una arquitectura tipo RiscV se recomienda familiarizarse con el conjunto de instrucciones que se espera utilizar según las necesidades del problema específico.

Durante la implementación de una arquitectura tipo RiscV, seccionar el diseño en módulos o bloques permite una más sencilla identificación de errores en el sistema permitiendo localizarlos a un bloque específico a la vez que facilita la organización del desarrollo.

Asimismo se destaca la importancia de una efectiva comunicación en equipo para una toma de decisiones grupal asertiva.

VI. BIBLIOGRAFÍA

[1] A. Waterman, K. Asanovic, Son, and R. Z. (2017). The RISC-V Instruction Set Manual v2.2. 2012 IEEE International Conference on Industrial Technology, ICIT 2012, Proceedings, I, 1–32. Extraído de: https://riscv.org//wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

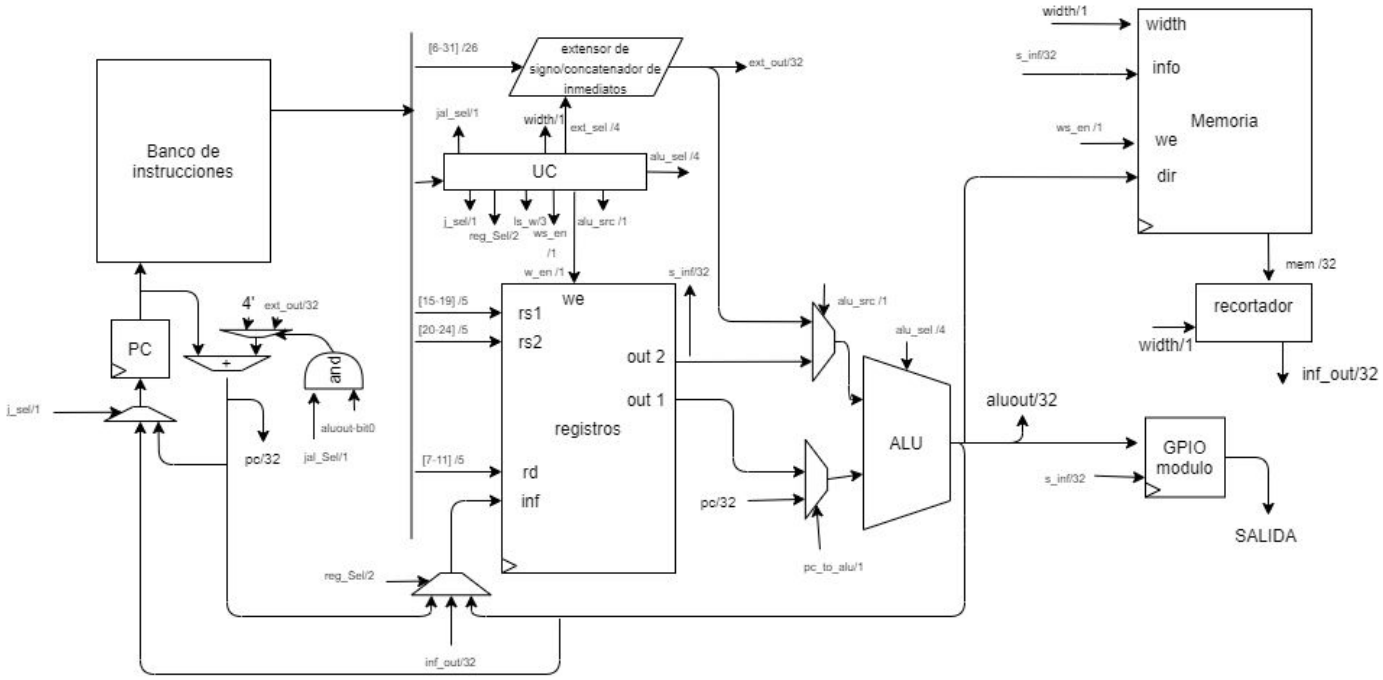
[2] D. Rojas (2018) Diseño de un ambiente de verificación basado en la metodología UVM para un microprocesador RISC-V 32I Instituto Tecnológico de Costa Rica, Escuela de Electromecánica. Extraído de: https://repositoriotec.tec.ac.cr/bitstream/handle/2238/10400/diseno_ambiente_verificacion_basado_metodologia_uvm_microprocesador_risc_v_231.pdf?sequence=1&isAllowed=y

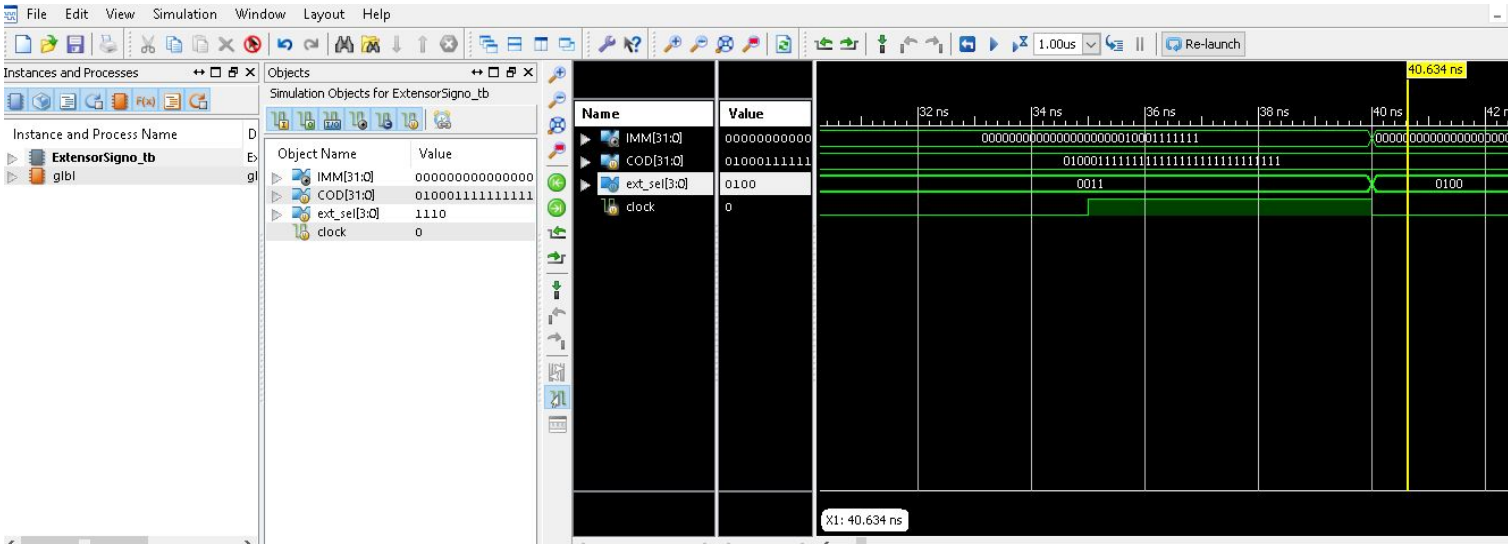
[3] Harris, S. L., & Harris, D. M. (2016). Digital design and computer architecture (ARM Edition ed.). Amsterdam [u.a.]: Elsevier, Morgan Kaufmann.

VII. Anexos

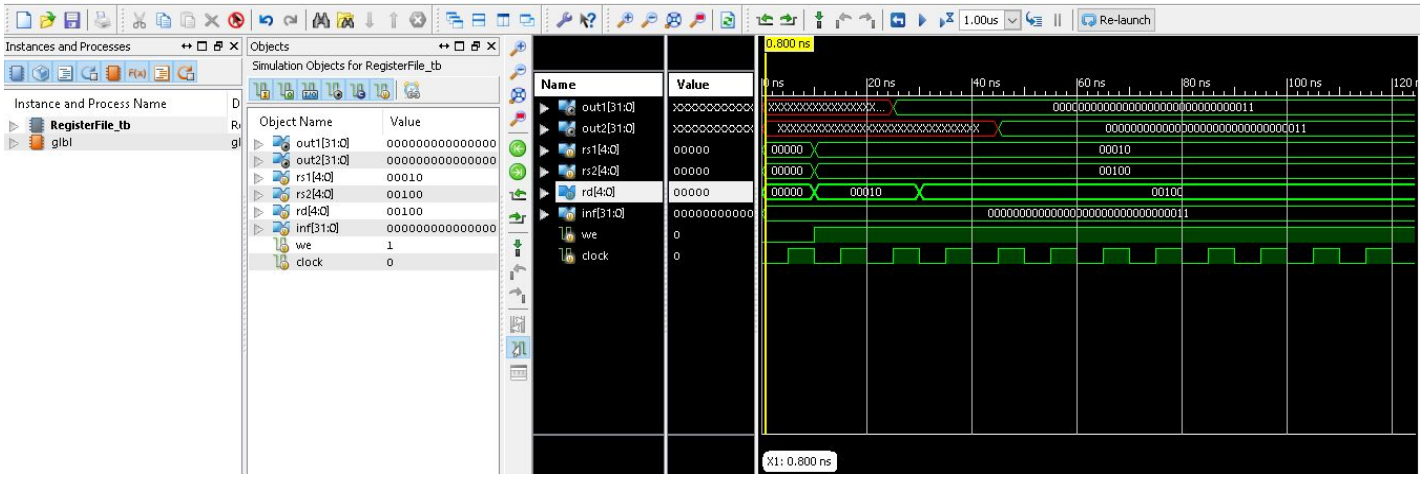
Link de repositorio de git: https://github.com/josedavid21/ProyectoRISCV-Mendez-Rojas-Solis-Soto

Diagrama final del circuito:

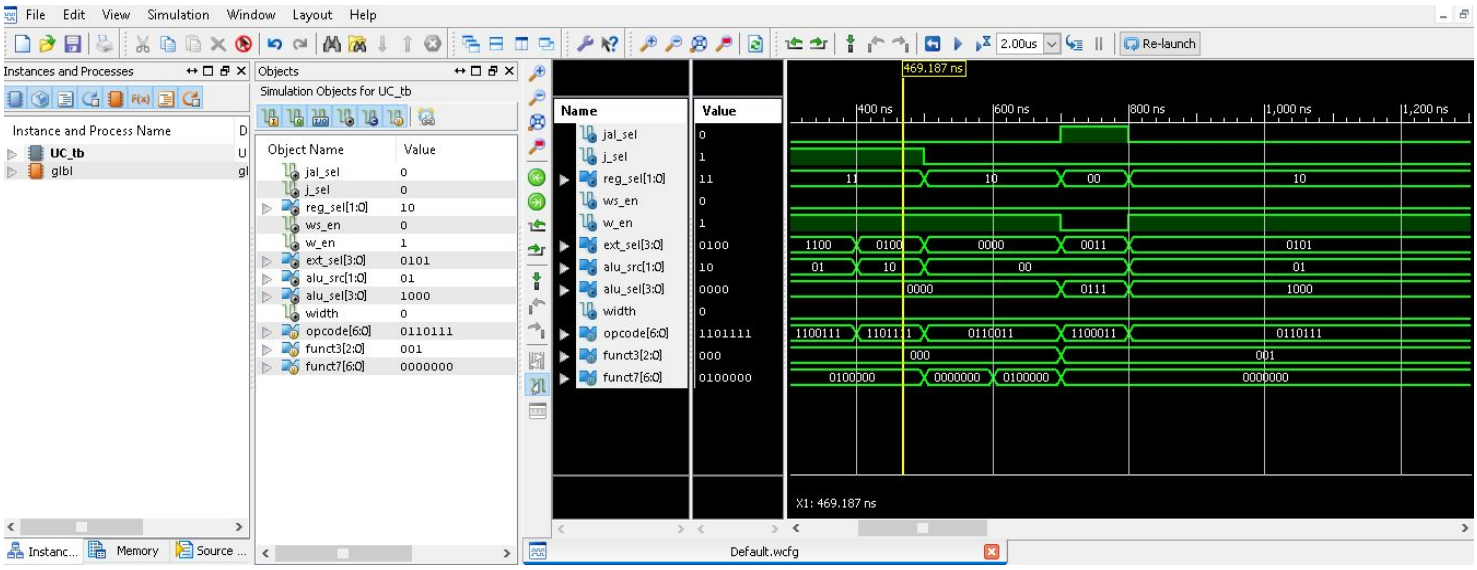




Tb del Extensor de Signo



Tb del Register File



Tb de la Unidad de Control