

# Implementación de un procesador RISC-V

Sebastián Nava López (ESCOM-IPN)



Noviembre 2022

# 1. Introducción

En este documento se encuentra la documentación de la implementación del procesador RISC-V bajo el set de instrucciones RV32I en dos tipos de ejecución: ciclo sencillo (*single-cycle*) y con pipeline. Adicionalmente se incluyen detalles sobre cada una de las unidades que forman la microarquitectura y se habla sobre la demostración desarrollada a partir de la implementación de este procesador. Mientras que la implementación con ciclo sencillo solo cubre las instrucciones de tipo I (excepto OP = 103), R, S/B y J, el diseño con pipeline cubre el total de las instrucciones de RV32I.

## 2. Simulación

El testbench del procesador puede ser corrido utilizando el comando *run* del Makefile incluido en el código de la implementación i.e. *make run*. Los parámetros para correr esta simulación se encuentran definidos en la constante GHDLRUNFLAGS.

## 3. Micro-arquitectura

### 3.1. Control de saltos

#### 3.1.1. Descripción

Esta unidad tiene el objetivo de controlar la señal *wpcSel* de tal forma que, dependiendo del tipo de instrucciones que se encuentren en las etapas de decodificación(IF/ID) y ejecución(ID/EX), así como el estado actual de la unidad de predicción (Unidad de predicción de saltos), se tome la siguiente instrucción de alguna de las cuatro señales disponibles en el multiplexor controlado por *wpcSel*. El algoritmo para elegir cada señal se encuentra descrito a continuación:

```
if jmpD:                                # Salto incondicional (jal,jalr) en IF/ID
    wpcSel := '00'                       # PC = PCtargetD
    flushD := '1'                       # Descartar la instrucción que pasa de FETCH -> IF/ID
if brD and takeD:                       # Salto condicional anticipado (instr. tipo B) cuando
    # el predictor (takeD) = '1'
    wpcSel := '00'                       # PC = PCtargetD
    flushD := '1'
if brE and (~takeE and brE):            # Salto condicional (instr. tipo B) en ID/EX donde
    # el predictor fue negativo (takeD = '0') en IF/ID
    # pero se cumple la condición de salto (breE = '1')
    wpcSel := '01'                       # PC = PCtargetE
    flushE := '1'                       # Descartar la instrucción que pasa de IF/ID -> ID/EX
    flushD := '1'
if brE and (takeE and ~brE):            # Salto condicional en ID/EX donde el predictor
    # fue '1' en IF/ID pero la condición de salto no se
    # cumple(breE = '0')
    wpcSel := '11'                       # PC = PCplus4E
    flushE := '1'
    flushD := '1'
else:                                    # Tomar la instrucción siguiente
```

wpcSel := '10'

# PC = PCplus4F

## 3.2. Unidad de Predicción de Saltos

### 3.2.1. Descripción

La unidad de predicción de saltos es una máquina de estados finitos basada en el diseño propuesto en el libro de Patterson y Hennessy mostrado en la figura 1. Esta unidad toma como entrada la señal `brc` de la unidad evaluadora de condiciones y genera la señal `take` que indica si la predicción de que la condición del salto condicional (instrucción tipo B) se va a cumplir o no.

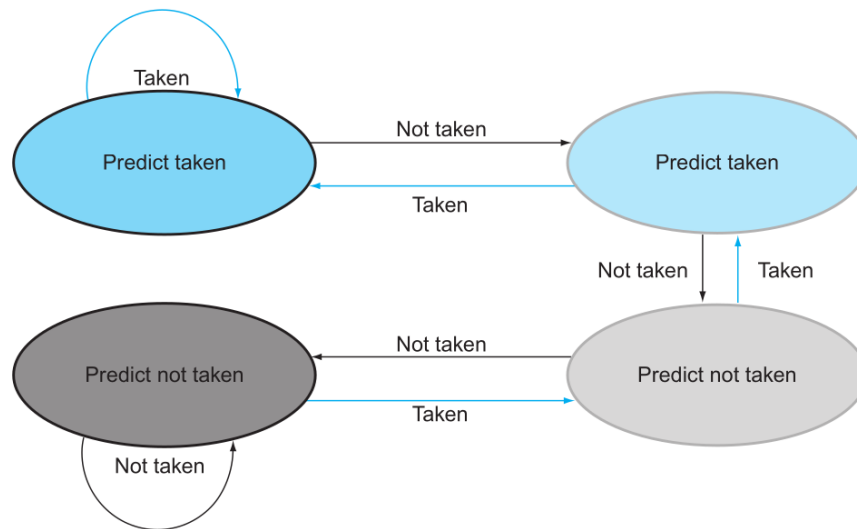


Figura 1: Máquina de estados finitos (FSM) de la unidad de predicción (tomado de *Computer Organization and Design RISC-V Edition: The Hardware Software Interface* pp. 314)

## 3.3. Unidad Aritmética-Lógica(ALU)

### 3.3.1. Descripción

Unidad para realizar operaciones lógicas y aritméticas que producen diferentes banderas dependiendo del resultado de la operación.

### 3.3.2. Tabla de control

(vease tabla 1).

## 3.4. Evaluador de condiciones

### 3.4.1. Descripción

Unidad genera la señal `brc` a partir de las señales recibidas de la ALU, donde el valor de `brc` indica que la condición se cumple o no a partir de los valores de las banderas. La condición que se evalúa depende de `funct3`.

aluOP	funct7 <sub>5</sub>	funct3	Operación	
00	x	x	ADD	$S = A + B$
01	x	x	SUB	$S = A - B$
10	x	000	ADD	$S = A + B$
11	0	000	ADD	$S = A + B$
	1	000	SUB	$S = A - B$
1x	x	001	SLL	$S = A \ll B$
	x	010	SLT	$S = A < B$
	x	011	SLTU	$S = A < B$
	x	100	XOR	$S = A \oplus B$
	0	101	SRL	$S = A \gg B$
	1	101	SRA	$S = A \gg B$
	x	110	OR	$S = A \vee B$
	x	111	AND	$S = A \wedge B$

Cuadro 1: Tabla de control para la ALU

### 3.4.2. Tabla de control

funct3	bre
000	$EQ = Z$
001	$NE = \neg Z$
100	$LT = N \oplus OV$
101	$GE = \neg LT$
110	$LTU = \neg C$
111	$GEU = C$
x	0



Cuadro 2: Tabla de control para el evaluador de condiciones

## 3.5. Archivo de Registros

### 3.5.1. Descripción

Unidad que contiene los 32 registros del procesador.

### 3.5.2. Tabla de control

CLR	CLK	WE	Operación
1	x	x	Reset
0		0	Banco' = Banco
0		1	Banco[A3] = WD3
x	x	x	RD1 = Banco[A1] RD2 = Banco[A2]


Cuadro 3: Tabla de control para el archivo de registros

## 3.6. Memoria de Datos (RAM)

### 3.6.1. Descripción

Unidad encargada del almacenamiento de datos con capacidad de lectura/escritura en byte, palabra(*word*) y media-palabra(*half-word*) con extensión de signo o padding de ceros en los modos de byte y media-palabra.

### 3.6.2. Tabla de control

CLK	WE	funct3	Operación
	1	000	bancoRAM[A] <sub>7:0</sub> = WD <sub>7:0</sub>
		001	bancoRAM[A] <sub>15:0</sub> = WD <sub>15:0</sub>
		x	bancoRAM[A] <sub>31:0</sub> = WD
x	0	000	RD = signExt(bancoRAM[Dir] <sub>7:0</sub> )
		100	RD = zeroExt(bancoRAM[Dir] <sub>7:0</sub> )
		001	RD = signExt(bancoRAM[Dir] <sub>15:0</sub> )
		101	RD = zeroExt(bancoRAM[Dir] <sub>15:0</sub> )
		x	RD = bancoRAM[Dir] <sub>31:0</sub>

Cuadro 4: Tabla de control para memoria RAM

## 3.7. Memoria de programa

### 3.7.1. Descripción

Existen dos unidades que pueden ser utilizadas como memoria de programa: *InstrMem* e *InstrMemProgrammable*, localizadas en el directorio *mem/instmem/*. La primera contiene diferentes programas que van desde pruebas simples de saltos, hasta implementaciones completas de algoritmos de ordenamiento como *bubble sort*. Estos programas están escritos a partir de la concatenación de diferentes constantes de tal forma que sea claro el formato de la instrucción. La segunda entidad tiene la capacidad de leer un programa desde un archivo de texto almacenado de forma local, en la computadora donde se lleve a cabo la simulación. La dirección de este archivo tiene que ser establecida dentro de la entidad y su formato es de una cadena en base hexadecimal de 32 bits por cada renglón. Un ejemplo de programa que puede ser leído por esta entidad se encuentra en el libro de Harris y Harris capítulo 7.6.3. *Testbench*

## 3.8. Pipelines

### 3.8.1. Descripción

Las pipelines de las diferentes etapas del procesador están diseñadas como una colección de registros con escritura y reset síncronos, de tal forma que exista cada uno de estos por cada señal que pase por el pipeline mas un registro adicional que almacena las señales de control que son pasadas a la siguiente etapa. Todas las entidades de este tipo se encuentran en el directorio *pipes/*.

## 4. Demo FPGA

Después de que se logró comprobar el funcionamiento del procesador a partir de los programas de prueba que fueron corridos por medio de un testbench y verificados a partir de los resultados de la simulación, se tomó la determinación de llevar las pruebas un paso más allá e implementar el procesador por medio de una tarjeta de desarrollo equipada con una FPGA, modelo Basys 3 de Digilent. Para esta implementación se diseñaron dos unidades para controlar el arreglo de 4 display de 8 segmentos y la cruceta de botones, ambos integrados a la tarjeta de desarrollo, de tal forma que el procesador pueda interactuar con ellos por medio de un esquema de mapeo de memoria. El mapa de memoria se muestra en la figura 2. La memoria de programa contiene un programa que evalúa si algún botón fue presionado y ejecuta diferentes acciones dependiendo del botón que fue presionado. Dicho programa hace referencia a alguna función que pueda tomar un solo número entero como argumento, y cuyo resultado pueda ser mostrado en el display. El archivo fuente de la memoria de programa cuenta con dos funciones que cumplen con esta característica: la suma del n-ésimo término, y el cálculo del n-ésimo término de la secuencia Fibonacci.

La forma en la que el usuario interactúa con la demostración es la siguiente: El display muestra una cuenta inicial (usualmente 0) que puede ser aumentada en una unidad con el botón superior (BTNU), y disminuida en una unidad con el botón inferior (BTND). Cuando se presiona el botón derecho (BTNR), se manda a llamar la función mencionada anteriormente y cuando la ejecución vuelve al programa principal, se muestra el resultado de esta función en el display. Finalmente, el switch SW0 y el botón central (BTNC) sirven para controlar la señal RESET. El diagrama de conexión entre el procesador (RV32I-core) y los otros periféricos se muestra en la figura 3

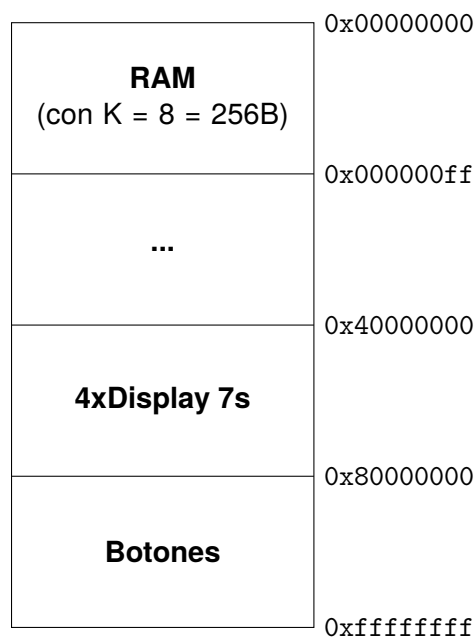


Figura 2: Mapa de memoria del diseño de demostración

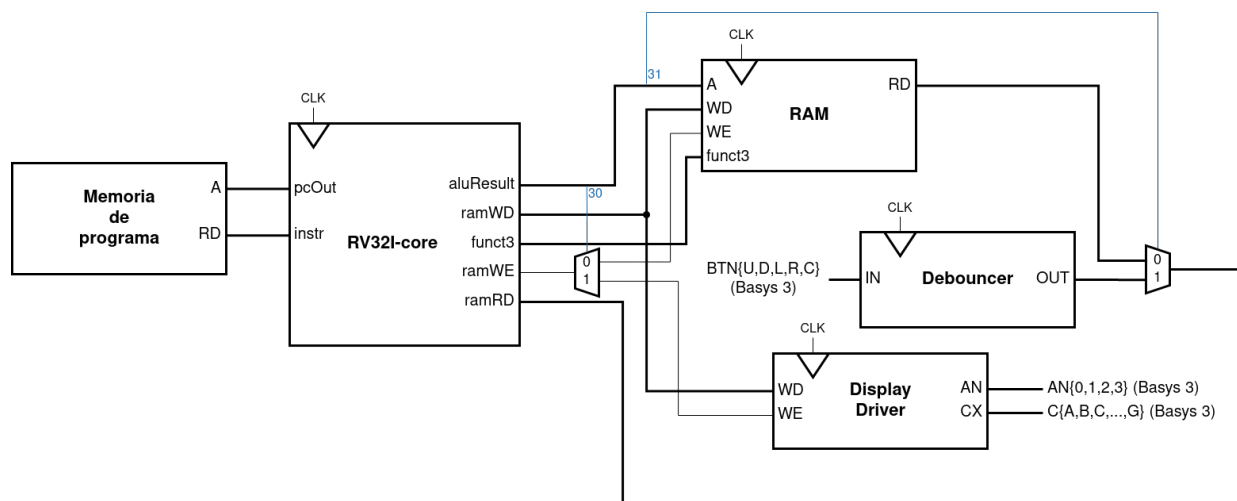
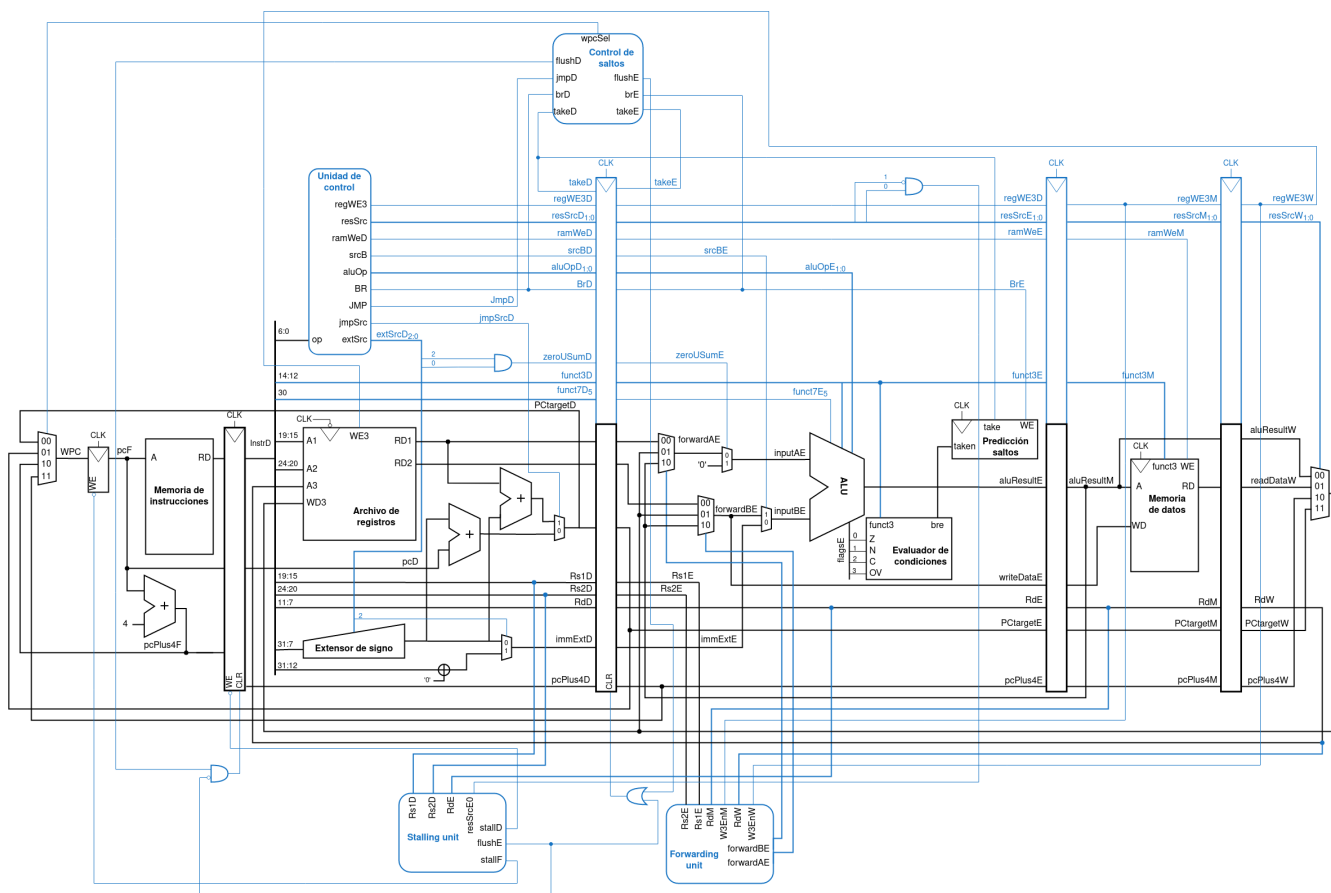


Figura 3: Diagrama general del diseño de demostración

## 5. Diagrama general del procesador





## Referencias

- [1] D.A. Patterson y J.L. Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2020. ISBN: 9780128203316.
- [2] S.L. Harris y D. Harris. *Digital Design and Computer Architecture, RISC-V Edition*. Elsevier Science, 2021. ISBN: 9780128200643.