UNIVERSIDAD AUTÓNOMA DE ZACATECAS

UNIDAD ACADÉMICA DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN ROBÓTICA Y MECATRÓNICA.

Nombre de la materia: Sistemas Digitales III.

Profesor: Dr. Remberto Sandoval Arechiga.

Trabajo: Reporte de microprocesador.

Nombres del equipo 6	Sergio Adad Bernal Adame Guillermo Cruz Fernández Gerardo Frías Donlucas.
Semestre y grupo	7º "B"
Fecha	03/12/2020

Resumen.

El proyecto consta del análisis, diseño y creación de una arquitectura Harvard Risc (figura 1) de 8 bits para un microprocesador el cual debe cumplir con un set de instrucciones que se estableció en un principio.

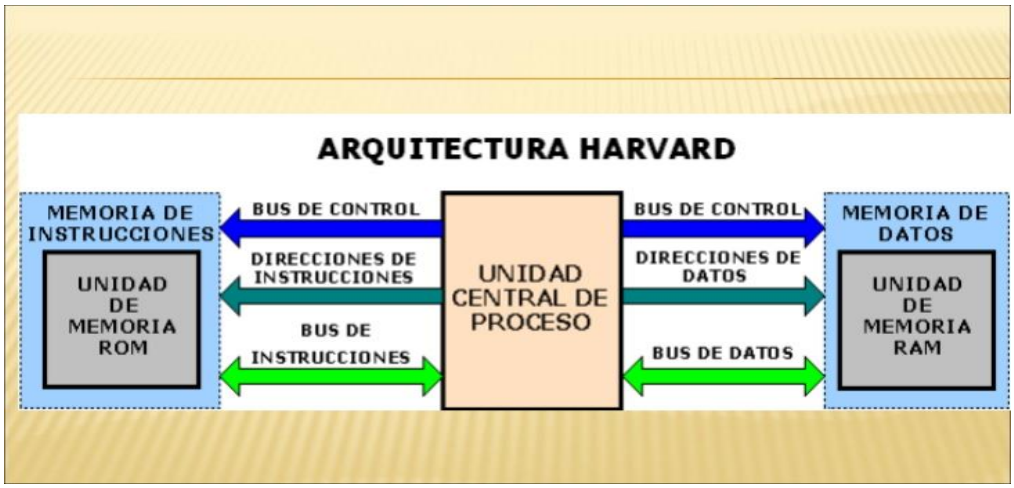

Figura 1. Arquitectura Harvard para un microcontrolador.

El microprocesador se le denomina al circuito electrónico que procesa la energía necesaria para el dispositivo electrónico en que se encuentra funcione, ejecutando los programas y comandos adecuadamente. La Unidad Centra de Procesos (CPU) de una computadora es un claro ejemplo de un microprocesador. Se denomina micro por su significado inglés que indica “pequeño”, en relación a la importancia de su función en un dispositivo, comparado a veces con el cerebro y con el corazón de los seres humanos. Este componente electrónico es el encargado de procesar y ejecutar las instrucciones codificadas en números binarios.

Para llevar a cabo este proyecto primero usando la metodología ya usada en clase, top down, se diseñó la arquitectura del microprocesador (esto incluye caja negra, caja blanca y submódulos), posteriormente se realizó la implementación o descripción en vivado, lenguaje verilog, y finalmente se realizaron las simulaciones correspondientes para observar el comportamiento y correcto funcionamiento de cada parte del microprocesador.

Índice.

1. Introducción

El microprocesador es un circuito capaz de recibir instrucciones, decodificarlas, buscar programas compatibles para ejecutarlas, ejecuta las instrucciones, analiza los datos y muestra los resultados de dicho proceso en cuestión de segundos (en tiempos muy cortos). Funcionan con la misma lógica que es usada por los CPU de una computadora digital, funcionan ejecutando operaciones lógicas y aritméticas simples como sumar, restar, multiplicar, etc.

Los microprocesadores se pueden distinguir por su velocidad interna y externa, que también determina los bits procesados por segundo, así como la capacidad de acceso a la memoria y el repertorio de instrucciones y programas a nivel informático que se pueden procesar.

Las principales partes que conforman un microprocesador(figura 2) son los registros, una unidad de control, una unidad aritmético lógica (ALU) y dependiendo del tipo de microprocesador también puede contener una unidad de cálculo en coma flotante. El microprocesador ejecuta las instrucciones almacenadas en números binarios organizados secuencialmente en la memoria principal y la ejecución de dichas se puede realizar mediante varias fases: prelectura de la instrucción desde la memoria principal (prefetch), envió de la instrucción a decodificador (Fetch), decodificación de la instrucción (determinar que instrucción y que se debe hacer), lectura de operando (si hay), escritura de los resultados en los registros.

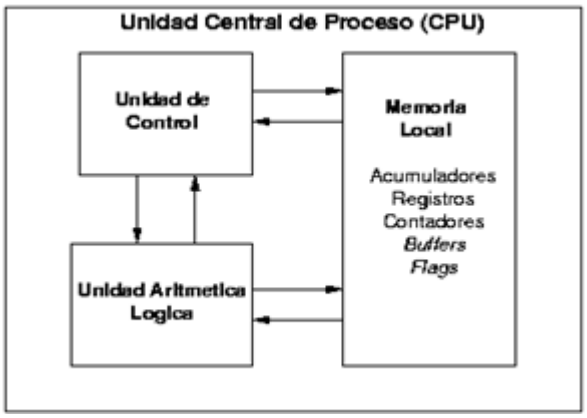

Figura 2. Diagrama que representa como se compone un microprocesador.

Cada una de las fases que se mencionaron anteriormente se realiza en uno o varios ciclos del CPU (figura 3). La duración de los ciclos depende o es determinada por la frecuencia del reloj, que genera pulsos a un ritmo constante de modo que genere varios pulsos o ciclos en segundos.

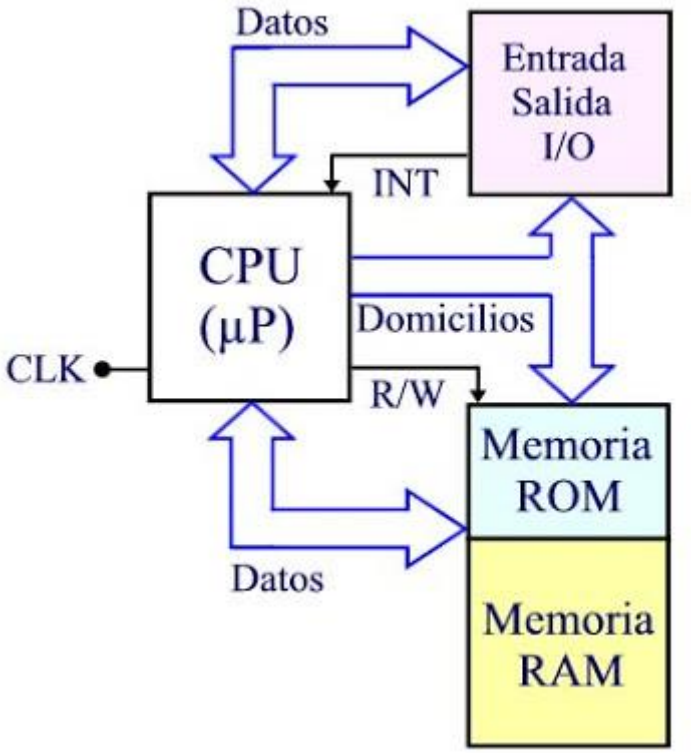

Figura 3. Partes de una computadora (CPU, Memorias y Puertos o Periféricos.)

Para poder hacer este proyecto o en cualquier otro proyecto de descripción de hardware correctamente y lograr lo que se propone se tiene que seguir una

metodología, que es de suma importancia para evitar caer en errores y ambigüedades.

Primero se diseña las caja negra y caja blanca de nuestro proyecto, así como declarar las entradas, salidas, el tamaño de bits de cada señal, también realizar algunas descripciones de lo que significa cada cosa para facilitar la comprensión y que posteriormente en el diseño de los submódulos no tengamos problemas.

Para finalizar esta parte se quiere mencionar que primero se hablara del diseño-arquitectura de la caja negra y caja blanca, posteriormente se dividió en submódulos donde que cada uno tiene una función específica para que en conjunto se logre el funcionamiento del proyecto, esto para poder trabajar conforme a la metodología que se vio en clase y así facilitar el trabajo.

2. Requerimientos.

Los requerimientos en el proyecto son el seguimiento de la metodología para diseñar las cajas negras y cajas blancas de cada módulo que conforma un microprocesador, esto tomando en cuenta lo visto en clase y lo que se investigó sobre microprocesadores, para darnos una idea de cómo poder diseñar el propio. Se tiene que tomar en cuenta los siguientes requisitos al momento de empezar el diseño del microprocesador:

- 8 registros, R0-R7.
- Registro PC.
- Operaciones enteras.

Instrucción	Argumentos	Descripción	Comentarios
LOAD	RX, #NUM	LOAD #Num to register X	#Num is 3 bits [0,7]
LOAD	RX, [RY]	LOAD data at address [RY] from memory	RY and RX are 3 bits [0,7]

STORE	#NUM	Store #Num to [RX] address memory	#Num is 3 bits [0,7]
STORE	[RX], RY	Store data at register RY in [RX] memory address	RY and RX are 3 bits [0,7]
MOVE	RX, RY	MOVE data from register RY to RX	RY and RX are 3 bits [0,7]
MATH	RX, OP	Do MATH operation with RX, and stores result in R0	OP: 0: R0=R0+RX 1: R0=R0-RX 2: R0=R0<<RX 3: R0=R0>>RY 4: R0=~RX 5: R0=R0&RX 6: R0=R0|RX 7: R0=R0^RX
JUMP	[RX], COND	JUMP PC to [RX] address if COND is true	COND: 0: NO CONDITION 1: No condition save PC in R7 2: Z flag is true 3: Z flag is false 4: C flag is true 5: C flag is false 6: N flag is true 7: N flag is false
NOP		NO OPERATION	

Tabla 1-. Set de instrucciones que se usara en el microprocesador.

Internamente se necesita un submódulo que permita hacer las operaciones lógicas y aritméticas, un bloque que sirva como control de las señales internas, otro bloque que nos ayude a saber que señal ira en que bus de salida, un bloque que sea donde se almacenen los datos y señales y uno que nos ayude a administrar las señales internas. Además, también se requerirán implementar una memoria RAM y una memoria ROM para poder conformar una computadora.

3. Arquitectura.

Microprocesador

El microprocesador consiste en un circuito que permite recibir, ejecutar, decodificar, analizar las señales y mostrar los resultados. Esto almacenando las instrucciones en números binarios organizados secuencialmente. Nos ayuda a que dependiendo de que instrucción (en binario) se le asigne o que queremos que haga, hará el proceso correspondiente y nos dará el resultado de dicho proceso.

En la siguiente imagen (figura 4) se puede observar todas las entradas y salidas necesarias para este proyecto.

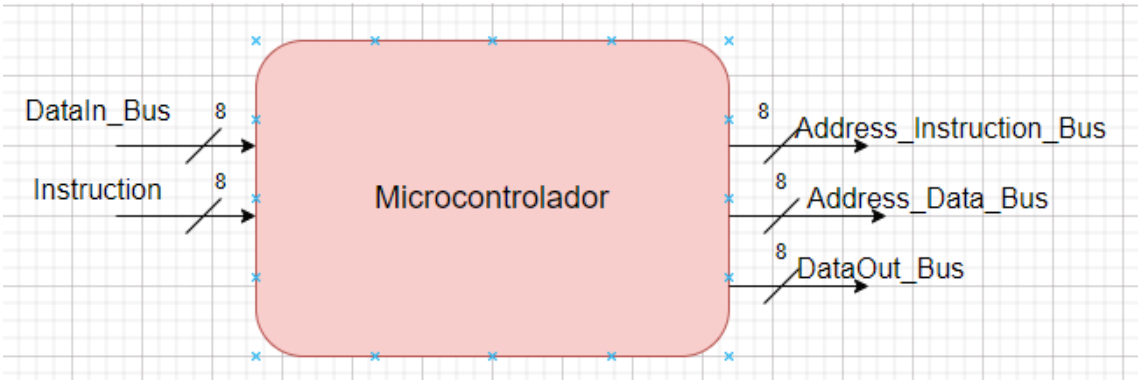

Figura 4. Caja negra de microcontrolador.

Para el correcto funcionamiento de este proyecto y como se había mencionado es necesario utilizar los submódulos que se muestran en la figura 5 ya que conforman lo que es el microprocesador y cada uno tiene una función específica para que el micro pueda realizar todo de manera adecuada, además de poder ver las conexiones internas entre cada bloque.

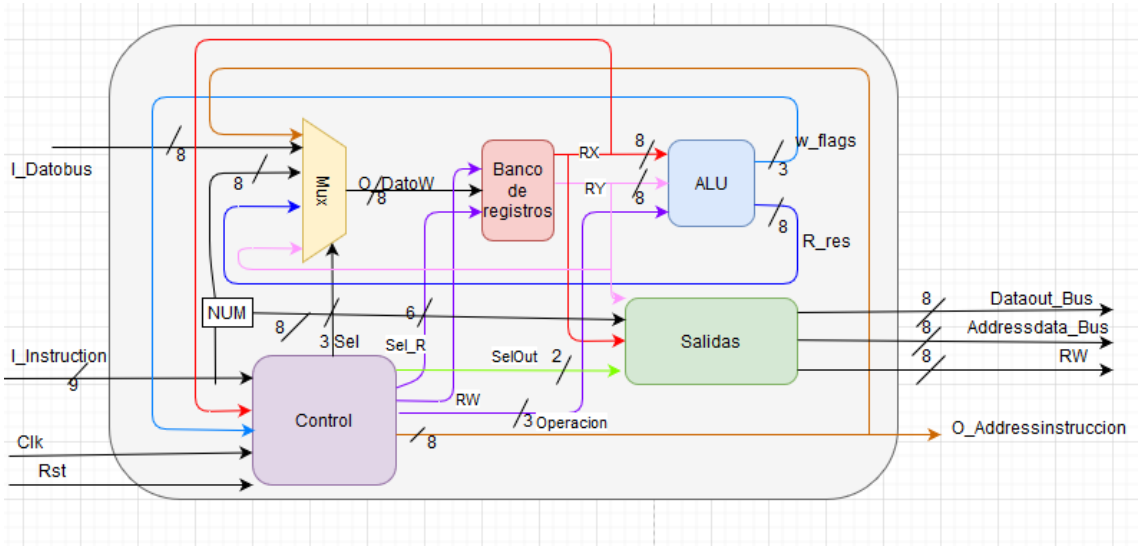

Figura 5. Caja blanca de microprocesador.

En la tabla siguiente (tabla 2) se muestra la descripción de entradas y salidas del microprocesador, así como también una explicación de su función.

Nombre de la señal	Dirección	Tamaño (bits)	Descripción
I_Datobus	Entrada	8	Señal de entrada de 8 bits del bus de datos.
I_Instruction	Entrada	9	Señal de entrada de 9 bits para el bus de instrucciones, nos indica que instrucción hacer.
Clk	Entrada	1	Señal de referencia temporal con una frecuencia de 100MHz.
Rst	Entrada	1	Señal que establece en el sistema un estado inicial.
Dataout_Bus	Salida	8	Es un dato de salida para cambiar una dirección de salida.
Addressdata_Bus	Salida	8	Dato de salida que dice la dirección de memoria que es necesaria para

			realizar una instrucción.
RW	Salida	8	Señal que indica si a leer o escribir(guardar) un dato de memoria.
O_Addressinstruccion	Salida	8	Dato de salida que dice la dirección de memoria que es necesaria para realizar una instrucción.

Tabla 2-. Tabla de descripción de entradas y salidas del microprocesador.

a) **ALU. Descripción:** El módulo ALU se encarga de hacer operaciones aritméticas y lógicas entre dos datos de 8 bits que se introducen desde los registros.

Se pueden realizar 8 operaciones u operandos: La suma, Resta, Corrimiento a la izquierda, corrimiento a la derecha, AND, OR, XOR y NOT. Para eso se usará la entrada "Operación" para seleccionar lo que deseamos hacer, así que de acuerdo a la tabla de abajo se seleccionará lo indicado. En la imagen 5 y tabla 3 se muestra las entradas y salidas de este bloque.

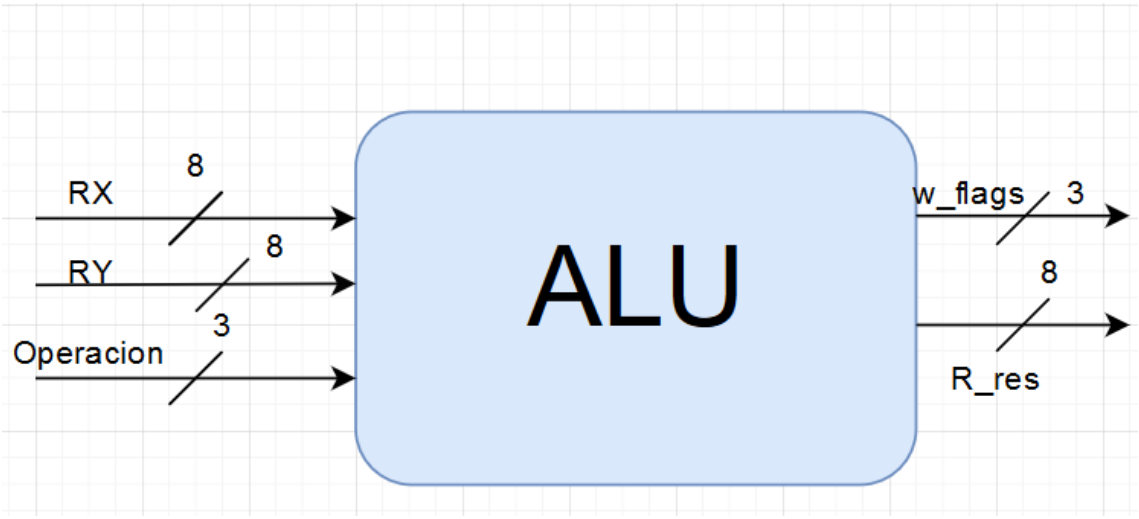

Figura 6. Caja negra del bloque ALU.

Señal	Dirección	Tamaño	Descripción
RX	Entrada	8	Dato de entrada de 8 bits, el cual representa uno de los dos datos entre los cuales se realiza la operación definida.
RY	Entrada	8	Dato de entrada de 8 bits el cual representa uno de los datos con los cuales se realizará una operación.
Operación	Entrada	3	Señal de bus que nos dice que instrucción se realizara
w_flags	Salida	3	Señal de bus que nos indica si se tiene que hacer un corrimiento, o alguna otra condición para poder realizar la operación.
R_res	Salida	8	Dato de salida de 8 bits el cual es el resultado de

			realizar una operación.

Tabla 3-. Tabla de descripción de entradas y salidas del ALU.

En la tabla 4 se muestra el funcionamiento de la ALU en las diferentes situaciones o casos que se pueden presentar y su representación en binario en donde se almacenara esa instrucción.

Nombre	Entrada de operación (DEC)	Entrada de operación (BIN)	Operación que se realiza.
Suma	0	000	R0= R0+RX
Resta	1	001	R0= R0-RX
Corrimiento izquierdo	2	010	R0= R0<<RX
Corrimiento derecho	3	011	R0= R0>>RX
AND	4	100	R0= R0&RX
OR	5	101	R0= R0|RX
NOT	6	110	R0= ~RX
XOR	7	111	R0= R0^RX

Tabla 4-. Descripción funcional del ALU.

En la tabla 5 se muestra cómo se controlará la selección de las condiciones de las banderas, estas condiciones son las que se pueden dar en una operación. La salida es de 3 bits para representar cada bandera.

w_flags	Flags	Descripción
000	Flags false	El resultado: No es 0, No es negativo, No tiene acarreo
001	Flag Z true	El resultado: Es 0, No es negativo, No tiene acarreo
010	Flag N true	El resultado: No es 0, Es negativo, No tiene acarreo
100	Flag C true	El resultado:

		No es 0, No es negativo, tiene acarreo

Tabla 5-. Descripción de las banderas en el ALU.

b) MUX. Descripción: Este bloque hace la función de seleccionar una de las diferentes entradas que tiene para poder mandarla a la salida y posteriormente que ingrese al banco de registros. En la imagen 6 se muestra el diseño de nuestra caja negra para este módulo, así como su descripción de las entradas y salidas en la tabla 6.

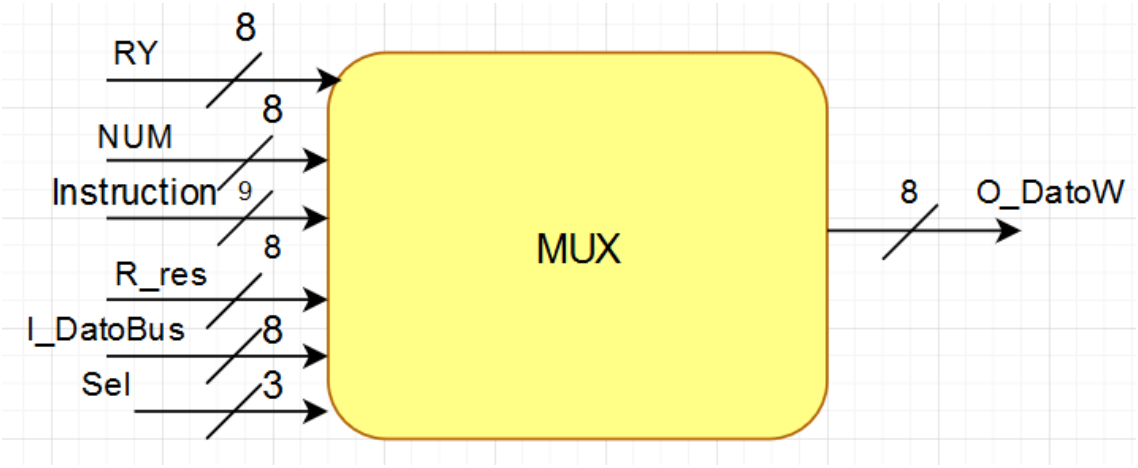

Figura 7. Caja negra del bloque MUX.

Sel	Dirección	Ancho	Descripción
Instruction	Entrada	9	Señal de entrada del bus de instrucción.
I_DatoBus	Entrada	8	Señal de entrada del bus de datos.
RY	Entrada	8	Registro que nos da un valor o número almacenado en el registro RY.
R_res	Entrada	8	Registro donde se almacena el resultado de las operaciones.

Sel	Entrada	3	Señal de entrada que ayuda a seleccionar que entrada se usara y mandarla al dato de salida.
Num	Entrada	8	Consiste en el valor del dato directo que corresponde a un número para las instrucciones que necesitan un valor directo.
O_DatoW	Salida	8	Es la señal que selecciona que dato vamos a usar para introducirlo al banco de registros.

Tabla 6-. Tabla de descripción de entradas y salidas del Multiplexor.

La tabla de a continuación (tabla 7) se muestra en que bits se asignara cada una de las señales para después poder hacer una selección y usarla en procesos posteriores.

Señal	Sel
0	000
Instruction	001
I_DatoBus	010
Num	011
RY	100
RX	101
R_res	110

Tabla 7-. Descripción de funcionamiento del MUX

c) **Banco de registros. Descripción:** Este bloque contiene 8 (R0-R7) registros que nos permitirán almacenar todos los datos que serán utilizados en mi microprocesador.

Permite leer y escribir en ellos para poder usar los datos deseados. En la figura 8 se muestra la caja negra del bloque de registros y en la tabla 8 explica o nos da una descripción de las entradas y salidas que tiene.

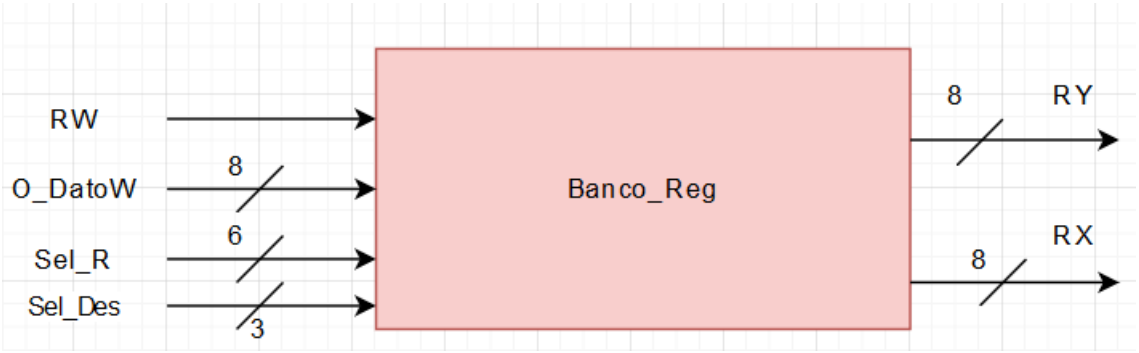

Figura 8. Caja negra de banco de registros.

Señal	Dirección	Tamaño	Descripción
RW	Entrada	1	Señal de entrada que nos indica si se va a escribir o a leer en los registros.
O_DatoW	Entrada	8	Señal de entrada que representa el dato seleccionado de una de las entradas del MUX.
Sel_R	Entrada	6	Señal de entrada que sirve para seleccionar el registro que se usara.
RY	Salida	8	Señal de salida de 8 bits que representa un dato almacenado en determinado registro para usarlo en alguna operación.
RX	Salida	8	Señal de salida de 8 bits que representa un dato almacenado

			en determinado registro para usarlo en alguna operación.
Sel_Des	Entrada	3	Señal que selecciona que dato se va a guardar en el banco de registros.

Tabla 8-. Descripción de entradas y salidas del banco de registros.

ReadWrite	RX	RY
0	R (Sel_R [2:1])	R (Sel_R [5:3])
1	R (Sel_R [2:1]) = DatoW	R (Sel_R [5:3])

Tabla 9-. Descripción del funcionamiento cuando ReadWrite toma valores de 1 y 0.

d) Salidas. Descripción: El Módulo de salidas nos ayuda a controlar las salidas que tendremos mediante la tabla 11. Este módulo nos ayuda a determinar, dependiendo de la instrucción, que valores se les asignaran a los buses de salida. En la figura 9 se ve el diseño de la caja negra para este bloque y en la tabla 10 la descripción de sus entradas y salidas.

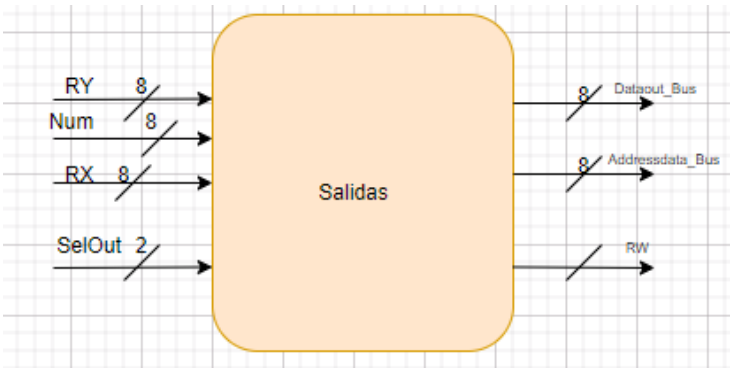

Figura 9. Caja negra del bloque salidas.

Señal	Dirección	Ancho	Descripción
NUM	Entrada	8	Señal de entrada para instrucción de un número inmediato.

RY	Entrada	8	Es un dato de entrada el cual se hará una operación definida.
RX	Entrada	8	Es un dato de entrada el cual se hará una operación definida.
SelOut	Entrada	2	Esta señal selecciona que operacion hará el bus de salida.
Dataout_Bus	Salidas	8	Es un dato de salida para cambiar una dirección de salida.
Addressdata_Bus	Salida	8	Dato de salida que dice la dirección de memoria que es necesaria para realizar una instrucción.
RW	Salida	1	Señal que indica si vamos a leer o guardar un dato de memoria.

Tabla 10-. Tablas de entradas y salidas del bloque de salidas.

Instruction	SelOut	Dataout_Bus	Addressdata_Bus	RW
NOP	00	0	0	0
LOAD. Dato de dirección RY.	01	0	RY	0
STORE. Num en la dirección RX.	10	Num	RX	1
STORE. RY en dirección RX.	11	RY	RX	1

Tabla 11-. Tabla de descripción funcional del bloque salidas.

e) **Control. Descripción:** Es el encargado de decidir:

¿Cuál dato se va a seleccionar?

¿Cuál operación va a hacer la ALU?

¿Cuáles registros vamos a usar?

Este módulo recibe la instrucción, la decodifica y es la encargada de decirle a los demás módulos que es lo que se tiene que hacer.

En la figura 10 se puede observar el diseño de la caja negra de nuestro bloque de control, además en la tabla 12 nos explica las entradas y salidas que tiene.

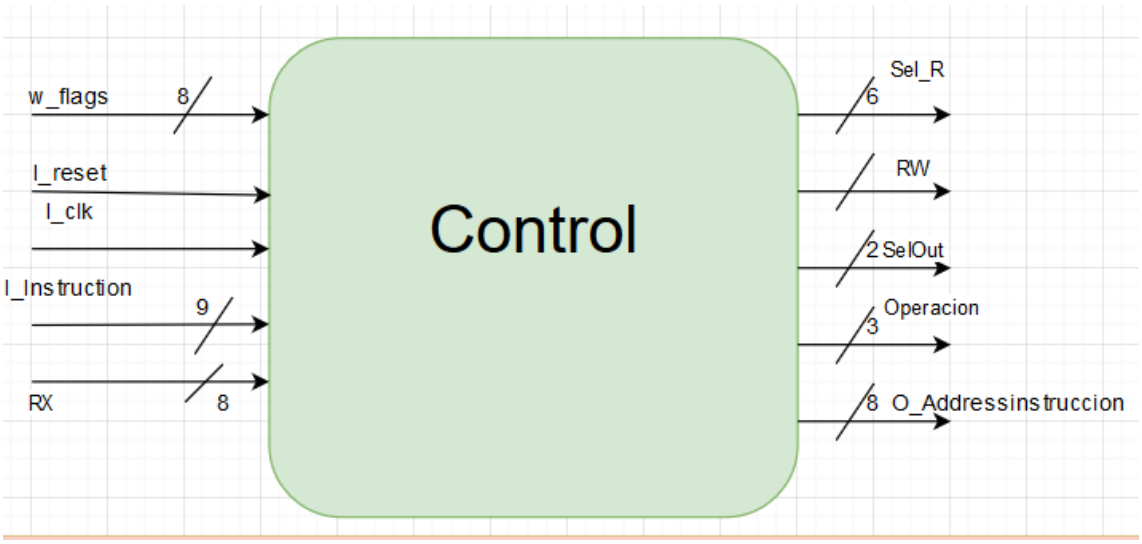

Figura 10. Caja negra del bloque Control.

Señal	Dirección	Ancho	Descripción
w_flags	Entrada	8	Señal de bus que nos indica si se tiene que hacer un corrimiento, o alguna otra condición para poder realizar la operación.
I_Instruction	Entrada	9	Señal de bus que nos indica que instrucción hacer.
Rst	Entrada	1	Señal que establece en el sistema un estado inicial.
Clk	Entrada	1	Señal de referencia temporal con una frecuencia de 100 MHz.
Sel_R	Salida	6	Señal encargada de seleccionar que registro (RX o RY)

			entrara al banco de registros.
RW	Salida	1	Señal de lectura y escritura en el banco de registros.
SelOut	Salida	2	Señal que controla la operación realiza el control de bus de salida.
Operación	Salida	3	Señal de bus que nos indica que operacion matemática se va a realizar, suma, resta, multiplicación/división.
O_Addressinstruccion	Salida	8	PC, Señal que tiene la dirección de la instrucción que se va a realizar.
RX	Entrada	8	Dato que contiene una dirección de registro para modificar el PC.

Tabla 12-. Descripción de entradas y salidas del bloque control.

	I_Instruction	Operacion	Sel_R	RW	SelOut	Sel	O_Address instruccion
Load	001, R_res, Num	0	000, RX	1	00	011	PC+1
Load	010, R_res, Num	0	RY, RX	1	01	010	PC+1
Store	011, R_res, Num	0	000, RX	0	10	000	PC+1
Store	100, RX, RY	0	RY, RX	0	11	000	PC+1
Move	101, RX, RY	0	RY, RX	1	00	101	PC+1
Math	110, RX, Operación	Operacion	000, RX	1	00	100	PC+1
Jump	111, RX, Cond	0	If Cond=1: a)000, R7 b)000, RX else 000, RX	If Cond=1: a)1 b)0 else 0	00	If Cond=1: 001 else 000	COND=0-->RX COND=1 a) PC+1 b) RX COND=2 -> if Z=1 -> RX else -> PC+1 COND=3 -> if Z=0 -> RX else -> PC+1 COND=4 if C=1 -> RX else -> PC+1

							COND=5 -> if C=0 ->RX else ->PC+1 COND=6 -> if N=1 ->RX else -> PC+1 COND=7 -> if N=0 -> RX else ->PC+1
Nop	0	0	0	0	00	000	PC+1

Tabla 13-. Descripción funcional del bloque de control.

4. Pruebas.

a) Microprocesador.

En esta parte se mostrará las simulaciones que se realizaron para cada módulo del microprocesador, con esto nos aseguramos de que el código que se implementó en verilog sea acorde o funcione de acuerdo a lo planeado. En la figura 11 se muestra la simulación del microprocesador al completo y en la figura 12 se muestra el microprocesador con las memorias, ya con todos los submódulos funcionando. En esta se puede observar primeramente los valores de las entradas y salidas del micro, además de mostrarnos los registros internos para visualizar los cambios que hay en los mismos y que se está haciendo.

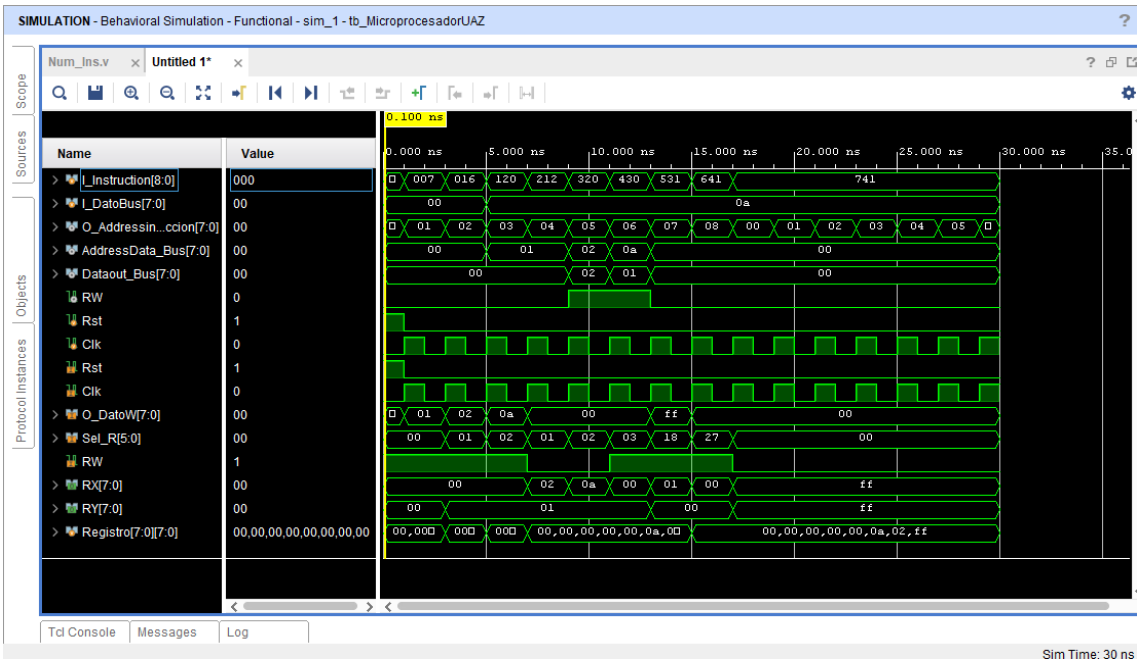

Figura 11. Simulación de microprocesador.

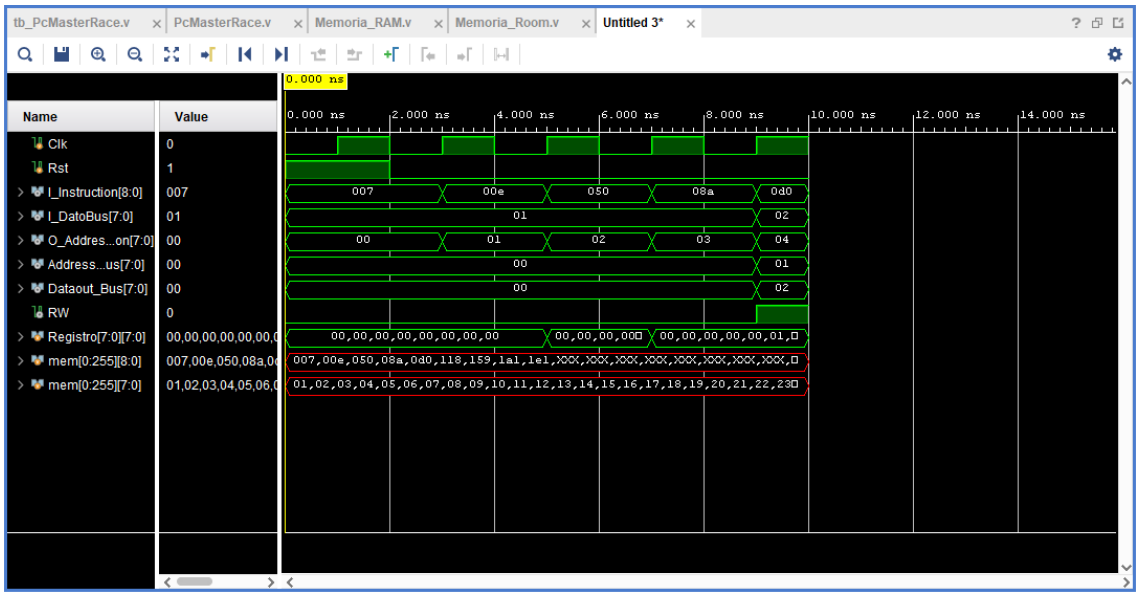

Figura 12. Simulación del microprocesador con memorias funcionando.

b) ALU

En esta parte se nos muestra como el módulo ALU (figura 13) corre las 8 operaciones, de esta manera nos aseguramos de que en efecto el ALU está trabajando correctamente.

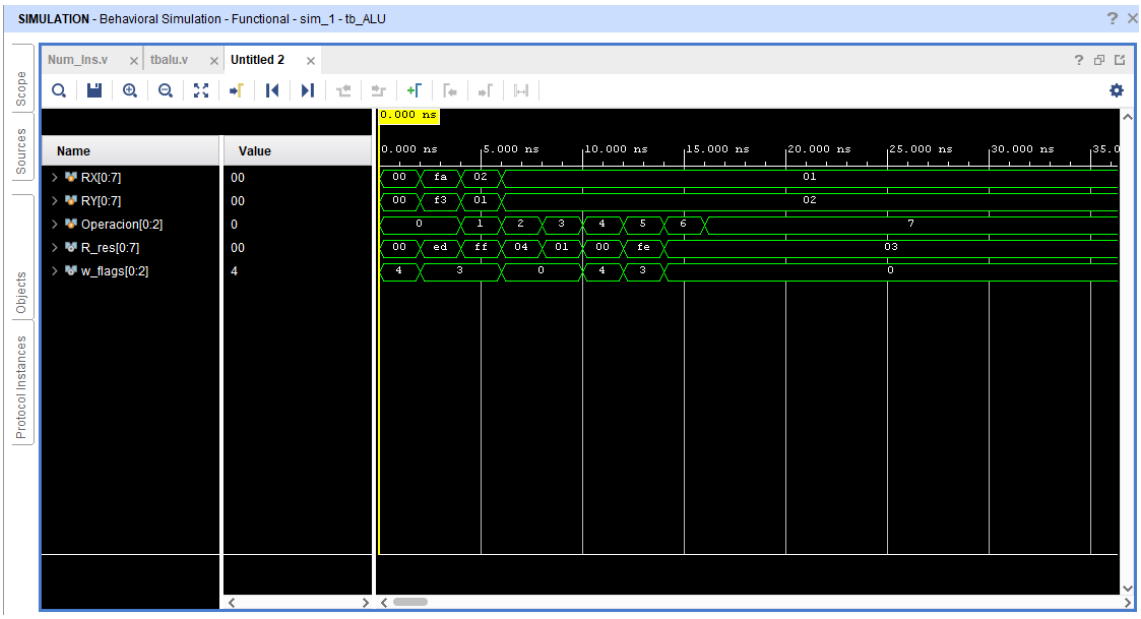

Figura 13. Simulación de ALU.

c) MUX.

A continuación, se ponen los resultados de nuestro MUX (figura 14) el cual logra ejecutar y funcionar correctamente.

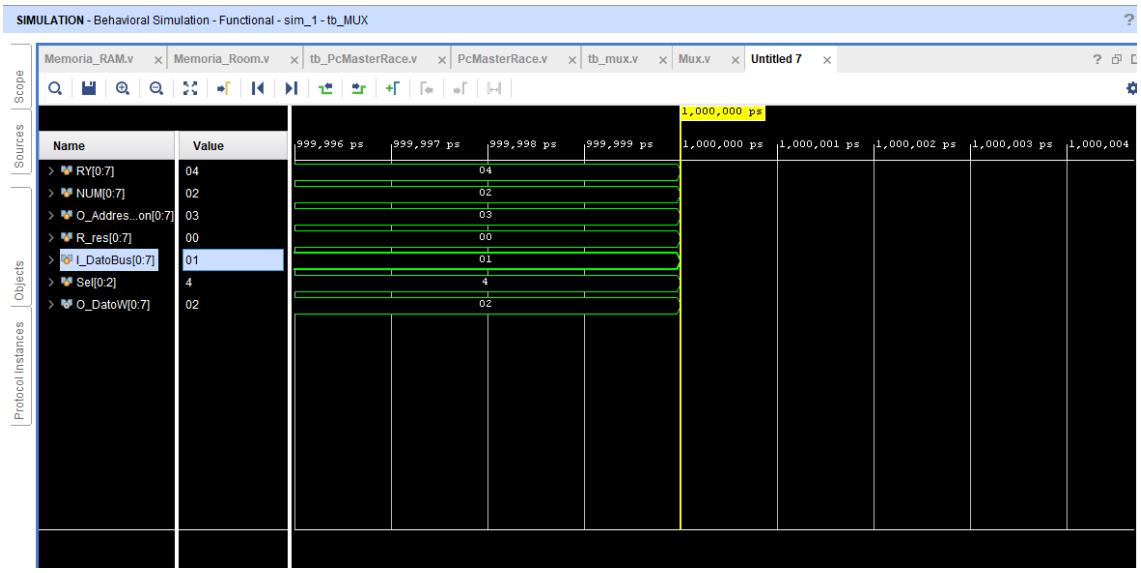

Figura 14. Simulación de MUX.

d) Banco de registros.

En esta sección (figura 15) se muestran los resultados del banco en el cual se da su correcto funcionamiento.

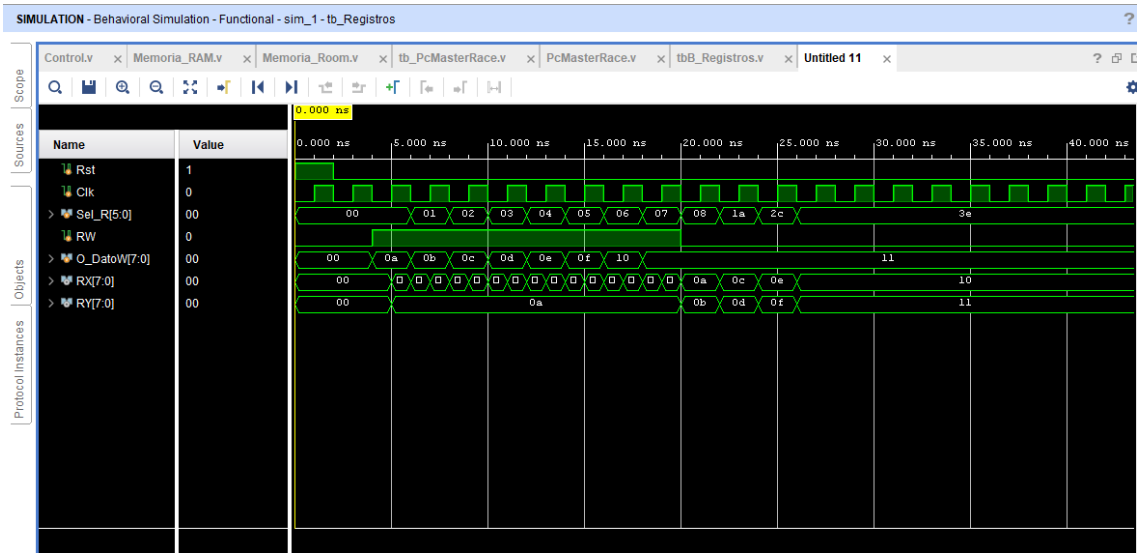

Figura 15. Simulación del banco de registros.

e) Salidas.

A continuación, se muestran los resultados del módulo de salidas (figura 16) el cual nos arroga su correcto funcionamiento.

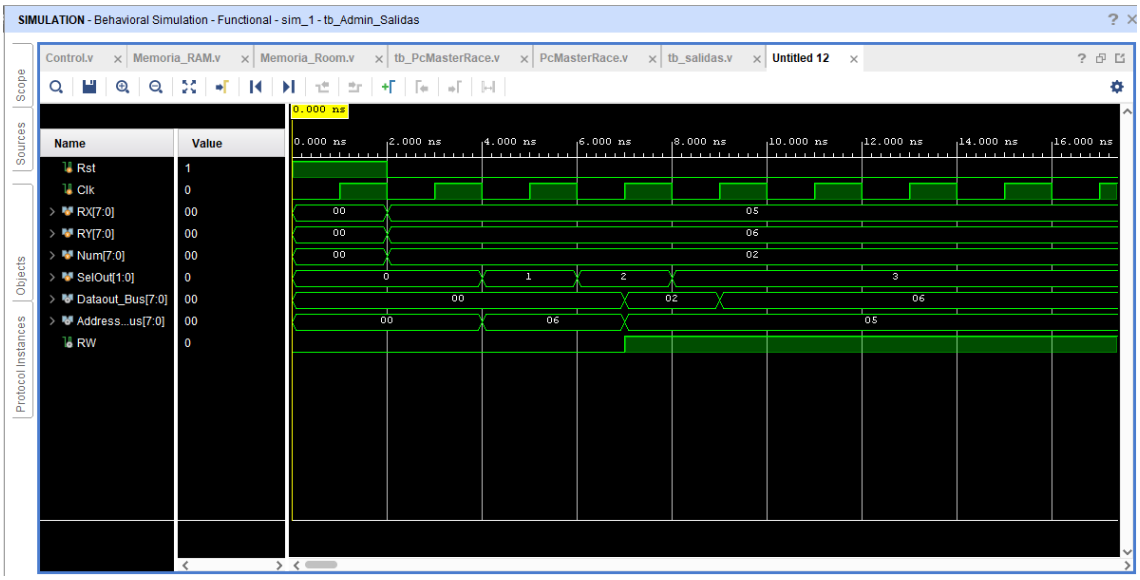

Figura 16. Simulación de Salidas.

f) Control

En este apartado se muestran los resultados del módulo de Control (figura 17) en donde todo funciona correctamente.

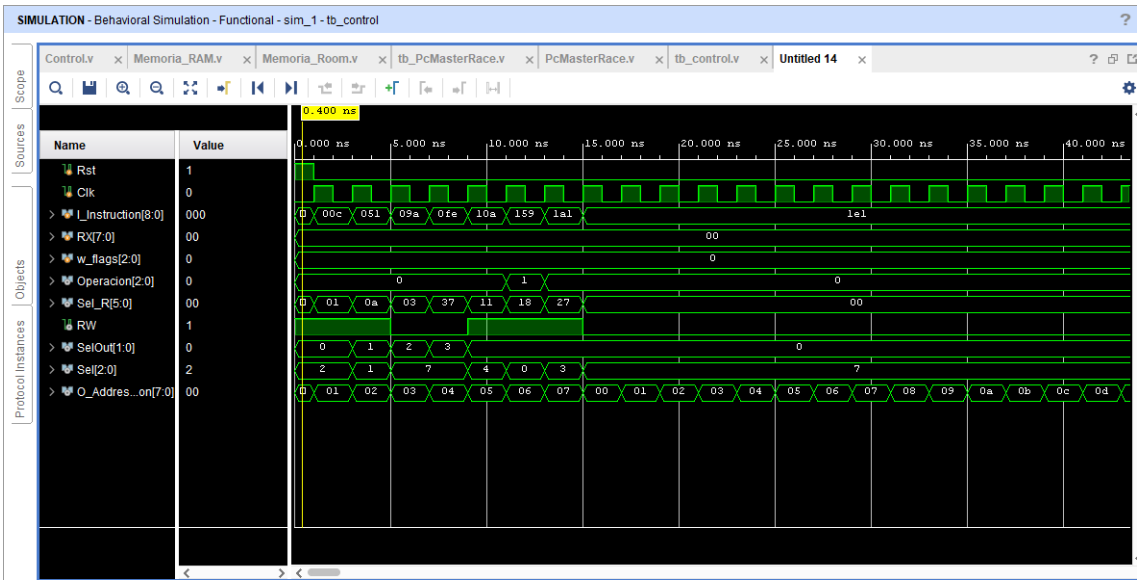

Figura 17. Simulación del módulo de control.

5. Análisis de resultados.

Con esto damos por sentado que nuestro microprocesador está funcionando correctamente, además de haber cumplido con los estándares establecidos para el mismo, el cual era tener el microprocesador funcional. Al conectar la memoria RAM y memoria ROM y comprobar que también hagan el funcionamiento correcto en conjunto se procede a hacer programas para hacer uso del mismo. Teniendo el micro y las memorias conectadas podemos decir que diseñamos nuestra primera computadora, a partir de un set de instrucciones y requisitos dados en el principio del proyecto.

Los microprocesadores interpretan el lenguaje máquina, pero las personas en realidad usan un lenguaje de programación de alto nivel (como C, C++) o un lenguaje ensamblador. Se creo un programa, basándose o apoyándose del set de instrucciones del micro, que pudiera hacer una operación de multiplicación (figura 18) y división (figura 19) de dos números, que posteriormente nos dará el resultado (guardado en un registro de la memoria).

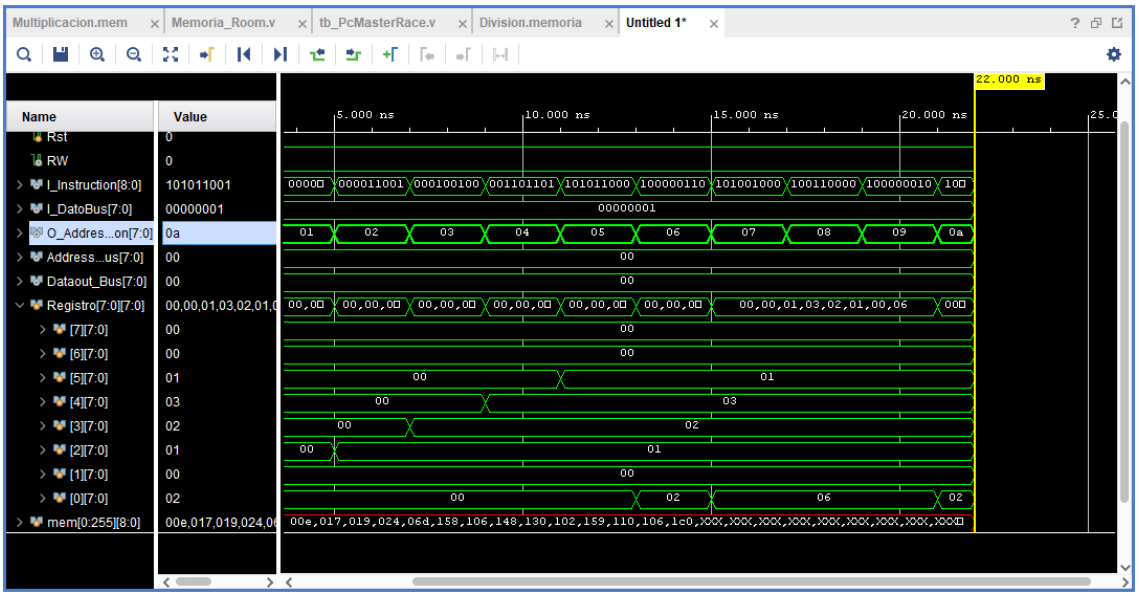

Figura 18. Simulación de la multiplicación en donde aparece la operación de 2x3 dando como resultado 6, el cual es guardado en un registro.

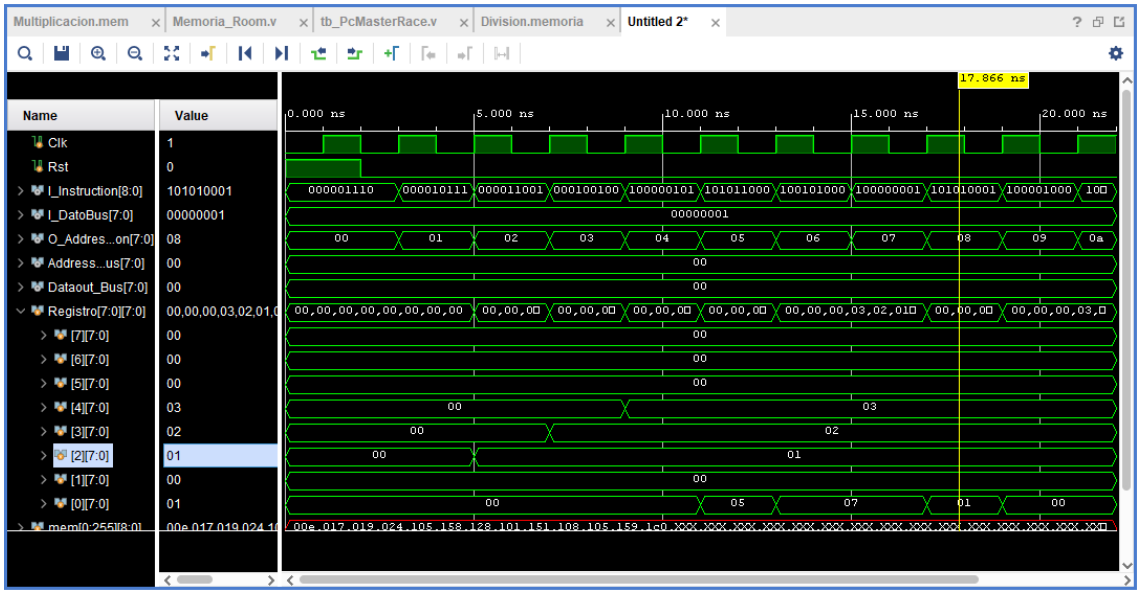

Figura 19. Simulación de la división, dando como operación 3 divido por 2, dando un 1 como resultado.

6. Conclusiones.

Si se sigue la metodología vista en clase el proceso por el que se llega al final de este proyecto se puede hacer más sencilla. Este microprocesador es sencillo a comparación de otros, que pueden llegar a implementarse más cosas dentro del mismo. Al momento de verlo en funcionamiento puede parecer que hace cosas sencillas como multiplicar o dividir, pero internamente es más complejo de lo que parece, la gente que no se adentra a este tipo de conocimientos no puede llegar a comprender que tan complejo puede llegar a ser y que partes conforman este tipo de circuitos, por más básicos que sean.

Hoy en día el tema de las computadoras y la digitalización es más normal, casi todo el mundo usa, en alguno de sus dispositivos, un microprocesador por lo cual estar involucrado en este tipo de proyectos es importante si se plantea seguir por este camino, ya que conforme el tiempo pasa la tecnología va cambiando/evolucionando. Para los dispositivos actuales los microprocesadores son una parte esencial porque con estos se pueden llegar a hacer cosas desde lo más simple hasta lo más complejo, es el corazón de los dispositivos actuales pueden hacer procesos muy complejos en poco tiempo, incluso hacer varios procesos a la vez.

En nuestra opinión este trabajo fue bastante entretenido ya que estuvimos trabajando en el varias semanas, en un principio nos costó un poco la parte del diseño de la arquitectura ya que no seguíamos la metodología y conforme se avanzó en el trabajo los problemas se iban reduciendo (en cuanto a atorarnos por no saber que seguía), la parte de implementar el código de verilog creo que fue la parte en la que avanzamos un poco más rápido ya que como nos menciona el profesor, si se sigue la mitología y se diseña la arquitectura bien estructurada al momento de seguir en los pasos posteriores reduces los errores y consigues terminar más rápido.

El proyecto (la clase) también incentivó el trabajar en grupos y tratar de sacar un trabajo entre todos, esto para tener la misma perspectiva que se tiene en grupos de trabajo laboral. Lo cual creo que fue algo problemático en un principio ya que surgen muchos problemas al no estar acostumbrados a hacer trabajos de tal magnitud durante varias semanas.

7. Referencias.

[1] Pertenece Samir Palnitkar (2003, February 21)), “Verilog HDL: A Guide to digital design and synthesis” Second Edition.

[2] Pertenece Stephen Brown and Zvonko Vranesic (-), “Fundamentals of Digital Logic with Verilog Desing” third edition.

[3] Pertenece JAMES E. PALMER DAVID E. PERLMAN (-), “INTRODUCCIÓN A LOS SISTEMAS DIGITALES”.

[4] Pertenece A.P.Godse D.A. Godse, “Microprocessors and Interfacing” First Edition 2009.

[5] Pertenece Muhammad Ali Mazidi Shujen Chen (+), “Freescale ARM Cortex-M Embedded Programming” 2014.

8. Apéndice.

Apéndice a.

Imagenes de referencia (figura 20), código de Deco y Jump, modulos empleados en la realización del modulo de control.

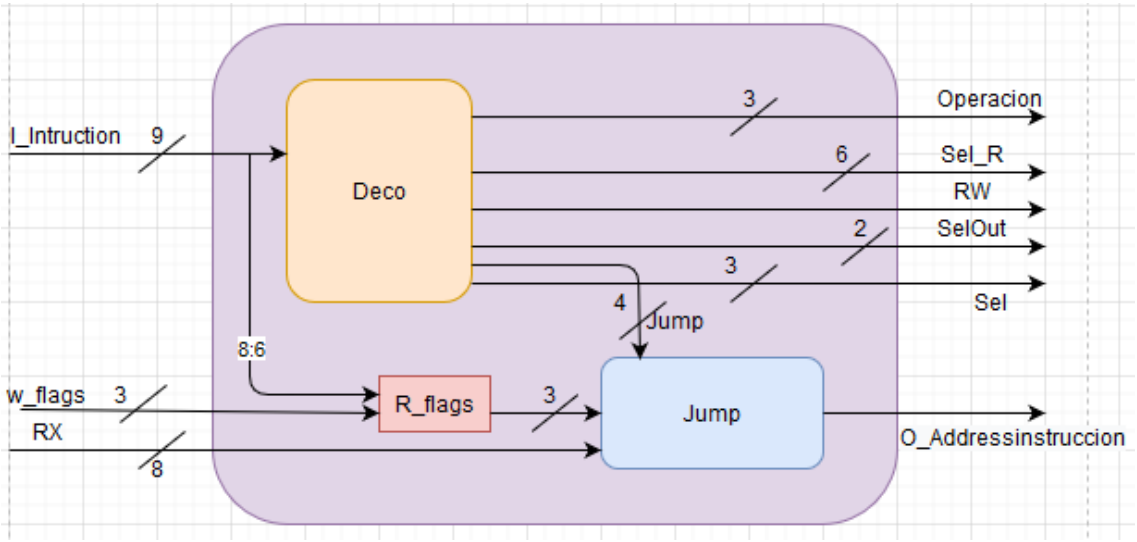

Figura 20. Módulos empleados en el módulo de control.

	I_Intruction	Operacion	Sel_R	ReadWrite	SelOut	Sel	Jump
LOAD	001,RX,NUM	0	000,RX	1	00	011	0001
LOAD	010,RX,NUM	0	RY,RX	1	01	010	0001
STORE	011,RX,NUM	0	000,RX	0	10	000	0001
STORE	100,,RX,RY	0	RY,RX	0	11	000	0001
MOVE	101,RX,RY	0	RY,RX	1	00	101	0001
MATH	110,RX,Operacion	Operacion	R0,RX	1	00	100	0001
JUMP	111,RX,COND	0	if COND=1 -> a)000,R7 b)000,RX else -> 000,RX	if COND=1 -> a) 1 b) 0 else -> 0	00	if COND=1 -> 001 else -> 000	{1´b1,COND}
NOP	0	0	0	0	00	000	0001

Figura 21. Tabla de referencia de las condiciones usadas en las instrucciones del módulo de control.

Código Deco.

```
module deco(

    input Rst,
    input Clk,
    input [8:0] I_Instruction,
    output reg [3:0] Jump,
    output reg [2:0] Operacion,
    output reg [5:0] Sel_R,
    output reg RW,
    output reg [1:0] SelOut,
    output reg [2:0] Sel
    );



    parameter //OpCode 3 bits mas significativos de la intruccion de
entrada: I_Intruction[8:6]
    LoadRxNum =3'b000,
    LoadRxARy =3'b001,
    StoreARxNum =3'b010,
    StoreARxRy =3'b011,
    MoveRxRy =3'b100,
    MathRxOp =3'b101,
    JumpRxCond =3'b110,
    NOP =3'b111;

    always @(I_Instruction) begin
        case (I_Instruction[8:6])
        LoadRxNum:begin   Jump<=4'b0001; Operacion<=0;
Sel_R<={3'b000,I_Instruction[5:3]}; RW<=1; SelOut<=0; Sel<=3'b010; end
        LoadRxARy:begin   Jump<=4'b0001; Operacion<=0;
Sel_R<={I_Instruction[2:0],I_Instruction[5:3]}; RW<=1; SelOut<=2'b01;
Sel<=3'b001; end
```

```
        StoreARxNum:begin Jump<=4'b0001; Operacion<=0;
Sel_R<={3'b000,I_Instruction[5:3]}; RW<=0; SelOut<=2'b10; Sel<=3'b111;
end
        StoreARxRy:begin  Jump<=4'b0001; Operacion<=0;
Sel_R<={I_Instruction[2:0],I_Instruction[5:3]}; RW<=0; SelOut<=2'b11;
Sel<=3'b111; end
        MoveRxRy:begin    Jump<=4'b0001; Operacion<=0;
Sel_R<={I_Instruction[2:0],I_Instruction[5:3]}; RW<=1; SelOut<=0;
Sel<=3'b100; end
        MathRxOp:begin    Jump<=4'b0001; Operacion<=I_Instruction[2:0];
Sel_R<={I_Instruction[5:3],3'b000}; RW<=1; SelOut<=0; Sel<=3'b000; end
        JumpRxCond:begin
           if(I_Instruction[2:0]==3'b001) // jump sin condicion y guardar pc en
R7
              begin  Jump<={1'b1,I_Instruction[2:0]}; Operacion<=0;
Sel_R<={I_Instruction[5:3],3'b111}; RW<=1; SelOut<=0; Sel<=3'b011;  end
           else //las demas condiciones de jump
              begin  Jump<={1'b1,I_Instruction[2:0]}; Operacion<=0;
Sel_R<={I_Instruction[5:3],3'b000}; RW<=0; SelOut<=0; Sel<=3'b111;  end
           end
        NOP:begin      Jump<=4'b0001; Operacion<=0; Sel_R<=0; RW<=0;
SelOut<=0; Sel<=3'b111; end
        default:begin  Jump<=4'b0001; Operacion<=0; Sel_R<=0; RW<=0;
SelOut<=0; Sel<=3'b111; end
        endcase
    end
endmodule
```

Código Jump.

```
module jump(

    input Rst,
    input Clk,
    input [3:0] Jump,
```

```
input [7:0] RX,
input [2:0] w_flags,
output [7:0] O_Addressinstruccion
);

reg [7:0] PC;

always @(posedge Clk, posedge Rst) begin
   if(Rst)
      PC<=0;
   else
      case (Jump)
         4'b0000:
            PC<=PC;
         4'b0001: // etapa de decodificacion de intruccion
            PC<=PC+1'b1;
         4'b1000:
            PC<=RX;
         4'b1001:
            PC<=RX;
         4'b1010:
            if(w_flags[0])
               PC<=RX;
            else
               PC<=PC+1'b1;
         4'b1011:
            if(~w_flags[0])
               PC<=RX;
            else
                  PC<=PC+1'b1;
         4'b1100:
            if(w_flags[2])
               PC<=RX;
```

```
            else
                PC<=PC+1'b1;
        4'b1101:
            if(~w_flags[2])
                PC<=RX;
            else
                PC<=PC+1'b1;
        4'b1110:
            if(w_flags[1])
                PC<=RX;
            else
                PC<=PC+1'b1;
        4'b1111:
            if(~w_flags[1])
                PC<=RX;
            else
                PC<=PC+1'b1;
        default: PC<=PC+1'b1;
    endcase
end
assign O_Addressinstruccion=PC;

endmodule
```

Apéndice b.

Modulo NUM (figura 22), encargado de codificar las señales de instrucción.

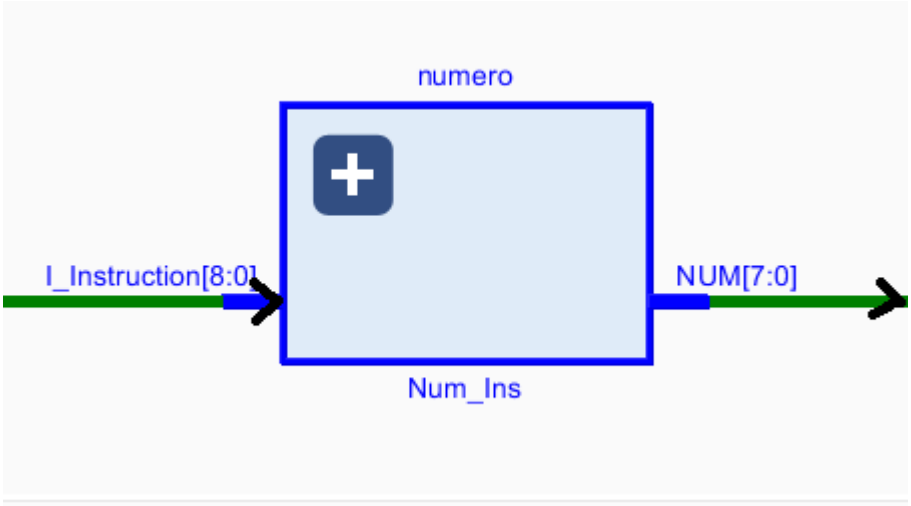

Figura 22. Imagen de referencia (extraída de vivado) módulo de NUM.

Apéndice C.

Memoria RAM y ROM.

Las memorias (figura 23) nos permiten almacenar información para usarlas en nuestros circuitos: datos, instrucciones, configuraciones, etc. Son los componentes esenciales para crear circuitos más complejos, como por ejemplo microprocesadores.

Las **memorias RAM** nos permiten almacenar datos y recuperarlos durante el funcionamiento del circuito. Son memorias donde podemos leer y escribir.

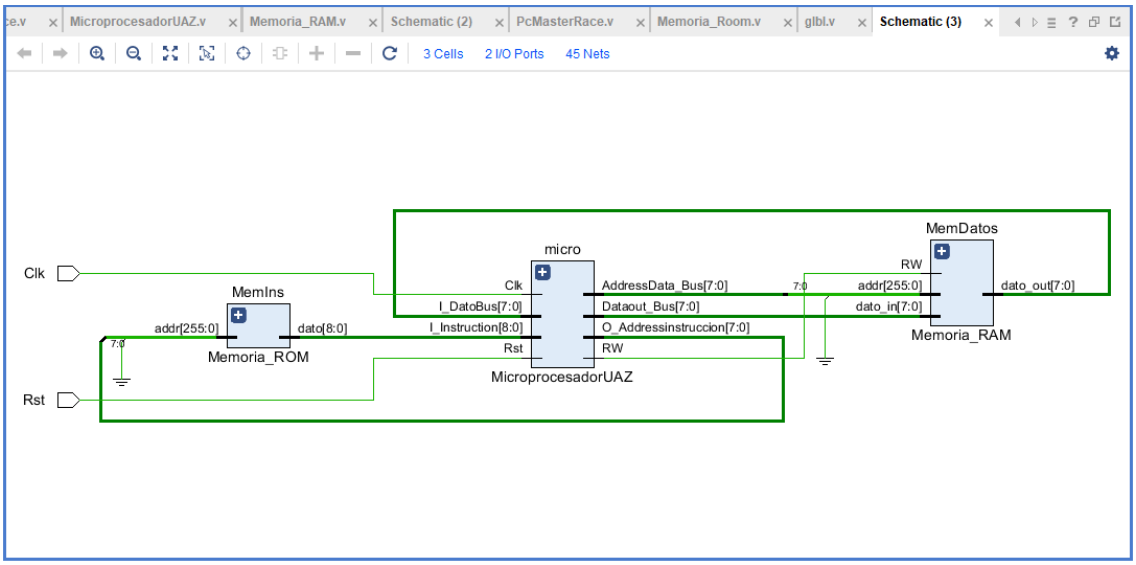

Figura 23. Imagen de referencia (extraída de vivado) microprocesador conectado a las memorias.

Apéndice D.

Multiplicación y división.

Tanto en la figura 24 como en la 25 se mostrará el procedimiento que toma realizar la multiplicación y división respectivamente.

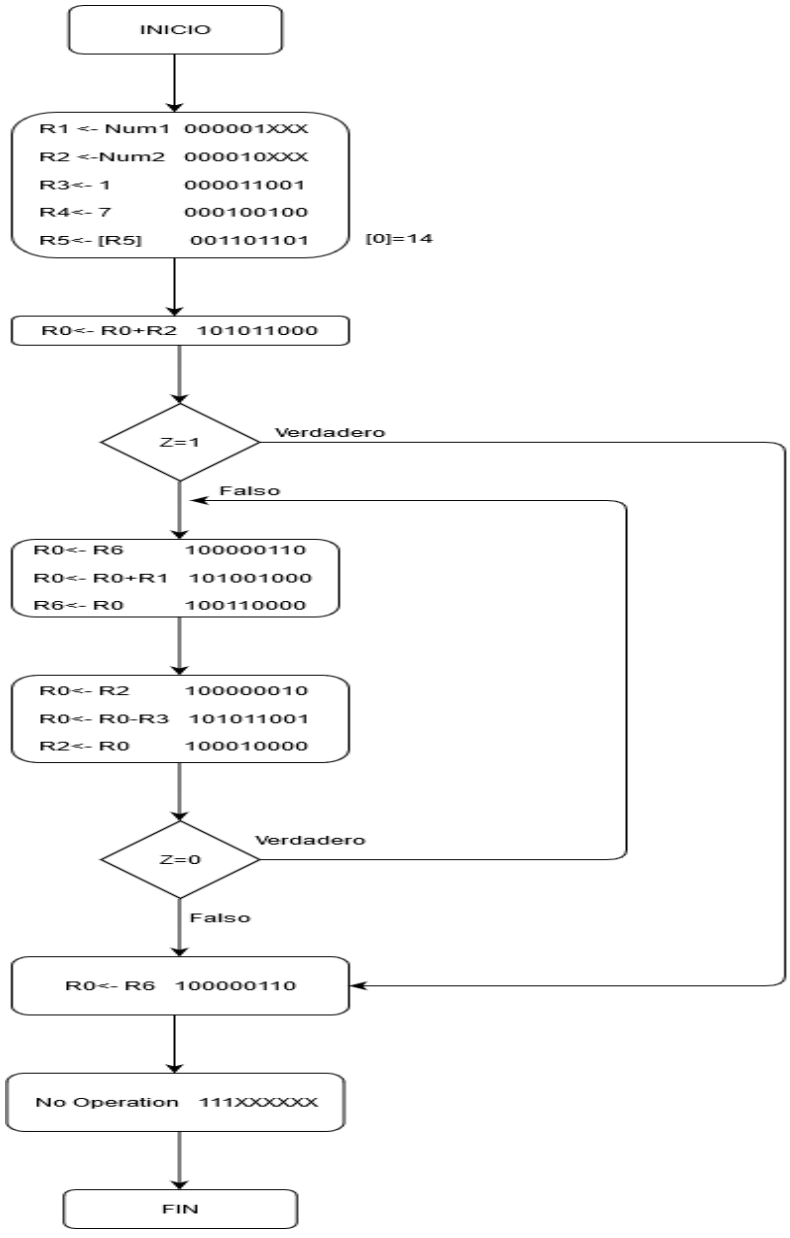

Figura 21. Procedimiento para la multiplicación.

Programa de multiplicación.

```
000001110
000010111
000011001
000100100
001101101
101011000
100000110
101001000
100110000
100000010
101011001
100010000
100000110
111000000
```

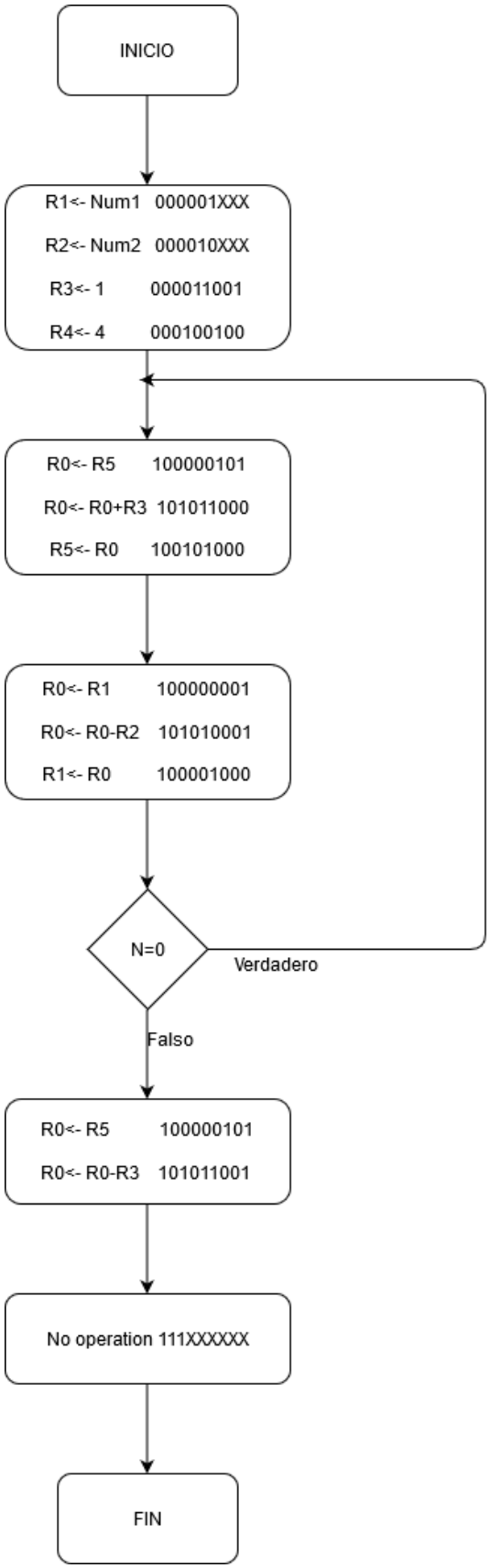

Figura 22. Procedimiento para realizar una división.

Programa de División.

```
000001110

000010111

000011001

000100100

100000101

101011000

100101000

100000001

101010001

100001000

100000101

101011001

111000000
```

9. Códigos de implementación.

a) Microprocesador.

```
module MicroprocesadorUAZ(
    input [8:0] I_Instruction,
    input [7:0] I_DatoBus,
    output [7:0] O_Addressinstruccion,
    output [7:0] AddressData_Bus,
    output [7:0] Dataout_Bus,
    output RW,
    input Rst,
    input Clk

    );
////////////////////////////////////////////////////////////////////////////////
wire [7:0]W_RX;
wire [7:0]W_RY;
wire [2:0]W_Operacion;
wire [7:0]W_R_res;
wire [2:0]W_W_flags;
wire [5:0]W_Sel_R;
wire W_RW;
wire [7:0]W_O_DatoW;
wire [2:0]W_Sel;
wire [7:0]W_NUM;
wire [1:0]W_SelOut;
////////////ALU////////////////////////////////////////////////////////////////
ALU
Operaciones(
    .RX(W_RX),
    .RY(W_RY),
    .Operacion(W_Operacion),
    .R_res(W_R_res),
    .w_flags(W_W_flags)
```

```
    );
//////////Salidas////////////////////////////////////////////////////////////
Admin_De_Salidas
Salidas(
    .Clk(Clk),
    .Rst(Rst),
    .RX(W_RX),
    .RY(W_RY),
    .SelOut(W_SelOut),
    .Num(W_NUM),
    .RW(RW),
    .AddressData_Bus(AddressData_Bus),
    .Dataout_Bus(Dataout_Bus)
);
//////////MUX/////////////////////////////////////////////////////////////////
MUX
Selector(
    .RY(W_RY),
    .NUM(W_NUM),
    .O_Addressinstruccion(O_Addressinstruccion),
    .R_res(W_R_res),
    .I_DatoBus(I_DatoBus),
    .Sel(W_Sel),
    .O_DatoW(W_O_DatoW)
);
//////////B_Registros/////////////////////////////////////////////////////////
B_Registros
Registros(
    .Rst(Rst),
    .Clk(Clk),
    .O_DatoW(W_O_DatoW),
    .Sel_R(W_Sel_R),
    .RW(W_RW),
```

```
    .RX(W_RX),
    .RY(W_RY)
);
//////////Control////////////////////////////////////////////////////
Control
Deco_ins(
    .Rst(Rst),
    .Clk(Clk),
    .I_Instruction(I_Instruction),
    .RX(W_R_res),
    .w_flags(W_W_flags),
    .Operacion(W_Operacion),
    .Sel_R(W_Sel_R),
    .RW(W_RW),
    .SelOut(W_SelOut),
    .Sel(W_Sel),
    .O_Addressinstruccion(O_Addressinstruccion)
);
//////////NUM////////////////////////////////////////////////
Num_Ins
numero(
    .I_Instruction(I_Instruction),
    .NUM(W_NUM)
);
 endmodule
```

b) ALU

```
module ALU(
input [0:7] RX,
input [0:7] RY,
input [0:2] Operacion,
```

```
output  [0:7] R_res,
output  [0:2] w_flags
    );
   /* assign R_res = R0;*/
   reg [8:0]R;
  always@(*)
       begin
          case(Operacion)

             0: R <= RY + RX;
             1: R <= RY - RX;
             2: R <= RY << RX;
             3: R <= RY >> RX;
             4: R <= RY & RX;
             5: R <= ~RX;
             6: R <= RY | RX;
             7: R <= RY ^ RX;

             endcase

                 end
                 assign R_res = R[7:0];
                 assign w_flags[0] = &(~R);
                 assign w_flags[1] = R[8];
                 assign w_flags[2] = R[7];

endmodule
```

c) MUX

```
module MUX(

input [0:7] RY,
```

```
input [0:7] NUM,
input [0:7] O_Addressinstruccion,
input [0:7] R_res,
input [0:7] I_DatoBus,
input [0:2] Sel,
output reg [0:7] O_DatoW
   );
   always@(*)
      begin
         case(Sel)

            0: O_DatoW <= R_res;
            1: O_DatoW <= I_DatoBus;
            2: O_DatoW <= O_Addressinstruccion;
            3: O_DatoW <= RY;
            4: O_DatoW <= NUM;
            default: O_DatoW <= 0;
         endcase
      end
endmodule
```

d) Banco de registros.

```
      module B_Registros(
input Rst,
input Clk,
input wire[7:0] O_DatoW,
input wire[5:0] Sel_R,
input wire RW,
//input wire [2:0] Sel_Des,
output [7:0] RX,
output [7:0] RY
);
```

```
reg [7:0] Registro[7:0];

always @(posedge Clk, posedge Rst) begin
if (Rst)
    begin
   Registro[0]<=0;
   Registro[1]<=0;
   Registro[2]<=0;
   Registro[3]<=0;
   Registro[4]<=0;
   Registro[5]<=0;
   Registro[6]<=0;
   Registro[7]<=0;
   end

   else
       if(RW)
           Registro[Sel_R[2:0]]<=O_DatoW;
   end
   assign RX=Registro[Sel_R[2:0]];
   assign RY=Registro[Sel_R[5:3]];
endmodule
```

e) Salidas.

```
module Admin_De_Salidas(

    input Clk,
    input Rst,
    input wire [7:0] RX,
    input wire [7:0] RY,
    input wire [1:0] SelOut,
```

```
input wire [7:0] Num,
output reg RW,
output reg [7:0] AddressData_Bus,
output reg [7:0] Dataout_Bus

);

 always @(posedge Clk, posedge Rst) begin
      if(Rst)
         begin
              Dataout_Bus<=0;
              AddressData_Bus<=0;
              RW<=0;
         end
        else
         case (SelOut)
         2'b00:
         begin
          Dataout_Bus<=0;
          AddressData_Bus<=0;
          RW<=0;
         end
         2'b01:
         begin
          Dataout_Bus<=0;
          AddressData_Bus<=RY;
         RW<=0;
         end
         2'b10:
         begin
         Dataout_Bus<=Num;
         AddressData_Bus<=RX;
         RW<=1;
```

```
                end
                2'b11:
                begin
                Dataout_Bus<=RY;
                AddressData_Bus<=RX;
                RW<=1;
                end
                endcase
                end
  always @(RX,RY,Num,SelOut)
     begin
        if(SelOut==2'b01)
           begin
           Dataout_Bus<=0;
           AddressData_Bus<=RY;
           end
        end
  endmodule
```

f) Control.

```
  module Control(
     input Rst,
     input Clk,
     input [8:0] I_Instruction,
     input [7:0] RX,
     input [2:0] w_flags,
     output [2:0] Operacion,
     output [5:0] Sel_R,
     output RW,
     output [1:0] SelOut,
     output [2:0] Sel,
     output [7:0] O_Addressinstruccion
```

```
);

wire [3:0] w_Jump;
reg [2:0] RFlags;

deco c_deco(
    .Rst(Rst),
    .Clk(Clk),
    .I_Instruction(I_Instruction),
    .Jump(w_Jump),
    .Operacion(Operacion),
    .Sel_R(Sel_R),
    .RW(RW),
    .SelOut(SelOut),
    .Sel(Sel)
    );

jump c_jump(
    .Rst(Rst),
    .Clk(Clk),
    .Jump(w_Jump),
    .RX(RX),
    .w_flags(w_flags),
    .O_Addressinstruccion(O_Addressinstruccion)
    );

always @(posedge Clk, posedge Rst) begin
    if(Rst)
        begin
            RFlags<=0;
        end
    else
        if (I_Instruction[8:6]==3'b101)
```

```
                RFlags<=w_flags;
        end
        endmodule
```

10. Códigos de test bench

a) Microprocesador.

```
module tb_MicroprocesadorUAZ;

reg [8:0] I_Instruction;
reg [7:0] I_DatoBus;
wire  [7:0] O_Addressinstruccion;
wire  [7:0] AddressData_Bus;
wire  [7:0] Dataout_Bus;
wire RW;
reg Rst;
reg Clk;

MicroprocesadorUAZ uut(
.I_Instruction(I_Instruction),
.I_DatoBus(I_DatoBus),
.O_Addressinstruccion(O_Addressinstruccion),
.AddressData_Bus(AddressData_Bus),
.Dataout_Bus(Dataout_Bus),
.RW(RW),
.Rst(Rst),
.Clk(Clk)

);

initial
   begin
```

```
        I_Instruction = 0;
        I_DatoBus = 0;
        Rst=1;
        Clk=0;
        #1 Rst=0;
            I_Instruction=9'b000_000_111; //load Rx Num
            #2 I_Instruction=9'b000_001_110; //load Rx Num
            #2 I_Instruction=9'b001_010_000; // load Rx [Ry]
              I_DatoBus=10;
            #2 I_Instruction=9'b010_001_010; // store [Rx] Num
            #2 I_Instruction=9'b011_010_000; // store [Rx] Ry
            #2 I_Instruction=9'b100_011_000; // move Rx Ry
            #2 I_Instruction=9'b101_011_001; // Math Rx OP
            #2 I_Instruction=9'b110_100_001; // Jump [Rx] Cond
            #2 I_Instruction=9'b111_100_001; // Nop
        end
    always
        begin
        #1 Clk = ~Clk;
        end
    endmodule
```

b) ALU.

```
    module tb_ALU;
    reg [0:7] RX;
    reg [0:7] RY;
    reg [0:2] Operacion;
    wire [0:7] R_res;
    wire [0:2] w_flags;

    ALU uut(
    .RX(RX),
```

```
.RY(RY),
.Operacion(Operacion),
.R_res(R_res),
.w_flags(w_flags)
);

initial
   begin
   RX =0;
   RY =0;
   Operacion =0;

   #2 RX= 8'b11111010; RY= 8'b11110011; Operacion= 0;
   #2 RX= 8'd2; RY= 8'd1; Operacion= 1;
   #2 RX= 8'd1; RY= 8'd2; Operacion= 2;
   #2 RX= 8'd1; RY= 8'd2; Operacion= 3;
   #2 RX= 8'd1; RY= 8'd2; Operacion= 4;
   #2 RX= 8'd1; RY= 8'd2; Operacion= 5;
   #2 RX= 8'd1; RY= 8'd2; Operacion= 6;
   #2 RX= 8'd1; RY= 8'd2; Operacion= 7;
      end
      endmodule
```

c) MUX.

```
module tb_MUX;
reg [0:7] RY;
reg [0:7] NUM;
reg [0:7] O_Addressinstruccion;
reg [0:7] R_res;
reg [0:7] I_DatoBus;
reg [0:2] Sel;
wire [0:7] O_DatoW;
```

```
MUX uut(
 .RY(RY),
 .NUM(NUM),
 .O_Addressinstruccion(O_Addressinstruccion),
 .R_res(R_res),
 .I_DatoBus(I_DatoBus),
 .Sel(Sel),
 .O_DatoW(O_DatoW)
);

initial
   begin
Sel = 3'd0;
R_res = 8'd0;
NUM = 8'd2;
I_DatoBus = 8'd1;
RY =8'd4;
O_Addressinstruccion = 8'd3;


#2
   Sel = 3'd1;
#2
   Sel = 3'd2;
#2
   Sel = 3'd3;
#2
   Sel = 3'd4;
```

```
    end
d)
e)  endmodule
```

f) Banco de registros.

```
module tb_Registros;

    reg Rst;
    reg Clk;
    reg [5:0]Sel_R;
    reg RW;
    reg [7:0] O_DatoW;
    //reg [2:0] Sel_Des;
    wire [7:0] RX;
    wire [7:0] RY;

 B_Registros uut(

    .Rst(Rst),
    .Clk(Clk),
    .Sel_R(Sel_R),
    .RW(RW),
    .O_DatoW(O_DatoW),
    //.Sel_Des(Sel_Des),
    .RX(RX),
    .RY(RY));

    initial
    begin

    Rst=1;
    Clk=0;
```

```
        Sel_R=0;
        RW=0;
        O_DatoW=0;

        #2 Rst=0; Sel_R=6'b000_000; RW=0; O_DatoW=8'b00000000;
        #2 Sel_R=6'b000000; RW=1;  O_DatoW=8'b00001010;
        #2 Sel_R=6'b000001; RW=1;  O_DatoW=8'b00001011;
        #2 Sel_R=6'b000010; RW=1;  O_DatoW=8'b00001100;
        #2 Sel_R=6'b000011; RW=1;  O_DatoW=8'b00001101;
        #2 Sel_R=6'b000100; RW=1;  O_DatoW=8'b00001110;
        #2 Sel_R=6'b000101; RW=1;  O_DatoW=8'b00001111;
        #2 Sel_R=6'b000110; RW=1;  O_DatoW=8'b00010000;
        #2 Sel_R=6'b000111; RW=1;  O_DatoW=8'b00010001;

        #2 Sel_R=6'b001000; RW=0;
        #2 Sel_R=6'b011010; RW=0;
        #2 Sel_R=6'b101100; RW=0;
        #2 Sel_R=6'b111110; RW=0;

        end
        always
        #1 Clk=!Clk;

    endmodule
```

g) Salidas.

```
    module tb_Admin_Salidas;
        reg Rst;
        reg Clk;
        reg [7:0] RX;
        reg [7:0] RY;
        reg [7:0] Num;
```

```
reg [1:0] SelOut;
wire [7:0] Dataout_Bus;
wire [7:0] AddressData_Bus;
wire RW;


Admin_De_Salidas uut(
.Rst(Rst),
.Clk(Clk),
.RX(RX),
.RY(RY),
.Num(Num),
.SelOut(SelOut),
.AddressData_Bus(AddressData_Bus),
.Dataout_Bus(Dataout_Bus),
.RW(RW)
);

initial
   begin
   Rst=1;
   Clk=0;
   RX=0;
   RY=0;
   Num=0;
   SelOut=0;

   #2 Rst=0; RX=8'd5; RY=8'd6; Num=8'd2; SelOut=0;
   #2 SelOut=2'b01;
   #2 SelOut=2'b10;
   #2 SelOut=2'b11;
   end
```

```
    always
        #1 Clk = !Clk;



endmodule
```

h) Control.

```
module tb_control;


    reg Rst;
    reg Clk;
    reg [8:0] I_Instruction;
    reg [7:0] RX;
    reg [2:0] w_flags;
    wire [2:0] Operacion;
    wire [5:0] Sel_R;
    wire RW;
    wire [1:0] SelOut;
    wire [2:0] Sel;
    wire [7:0] O_Addressinstruccion;

    Control uut(
    .Rst(Rst),
    .Clk(Clk),
    .I_Instruction(I_Instruction),
    .RX(RX),
    .w_flags(w_flags),
    .Operacion(Operacion),
    .Sel_R(Sel_R),
    .RW(RW),
    .SelOut(SelOut),
    .Sel(Sel),
```

```
        .O_Addressinstruccion(O_Addressinstruccion)
        );

        initial
            begin
            Rst=1;
            Clk=0;
            I_Instruction=0;
            RX=0;
            w_flags=0;

            #1 Rst=0; I_Instruction=9'b000_001_100; //load Rx Num
            #2 I_Instruction=9'b001_010_001; // load Rx [Ry]
            #2 I_Instruction=9'b010_011_010; // store [Rx] Num
            #2 I_Instruction=9'b011_111_110; // store [Rx] Ry
            #2 I_Instruction=9'b100_001_010; // move Rx Ry
            #2 I_Instruction=9'b101_011_001; // Math Rx OP
            #2 I_Instruction=9'b110_100_001; // Jump [Rx] Cond
            #2 I_Instruction=9'b111_100_001; // Nop

            end

        always
            #1 Clk = !Clk;
    endmodule
```