

UNIVERSIDAD AUTÓNOMA DE ZACATECAS
“Francisco García Salinas”
UNIDAD ACADÉMICA DE INGENIERÍA ELÉCTRICA



**PROGRAMA ACADÉMICO: INGENIERÍA EN ROBÓTICA Y
MECATRÓNICA**
SISTEMAS DIGITALES III

Docente: Dr. Remberto Sandoval Arechiga

Microprocesador de 8 bits con arquitectura Harvard

Equipo 4, Robóticos:

Nelson Eduardo Coronado Gamez

Clara Verónica Guerrero Correa

Hilda Berenice Espinosa Herrera

7“B”

Fecha: 30/11/2020

Resumen

En este documento se encontrará como a partir de un set de instrucciones y una caja negra se hizo un microprocesador de 8 bits tipo RISC y con arquitectura Harvard. Este microprocesador de acuerdo a su set de instrucciones se fue viendo cuáles componentes eran necesarios para su correcto funcionamiento.

Se podrá encontrar las cajas negras de los componentes del Microprocesador, la descripción de su funcionamiento, su código de implementación, y las simulaciones que comprueban el funcionamiento de cada uno de ellos.

Por último también podrá encontrar que a través del microprocesador se pueden realizar las operaciones de multiplicación y división, esto es a través de las instrucciones del microprocesador, mediante un set de instrucciones para cada operación antes mencionada.

Índice

Introducción.....	3
Requerimientos.....	3
Arquitectura.....	4
Implementación.....	16
Pruebas.....	17
Análisis de resultados.....	21
Conclusiones.....	22
Referencias.....	22
Apéndice	23
Apéndice A. Código de implementación	23
Apéndice B. Código de tests benches	43

Introducción

Un microprocesador es la unidad central de procesamiento, este es el encargado de controlar todo un sistema por medio de la decodificación y ejecución de un set de instrucciones (conjunto de instrucciones) que se encuentra ya dado.

Existen varios tipos de Microprocesadores con diferentes arquitectura, en este caso se va a realizar un Microprocesador tipo RISC con arquitectura Harvard. Que sea de tipo RISC quiere decir que las instrucciones son de tamaño fijo, que están reducidas lo más posible y que solo se accede a memoria cuando hay una instrucción de carga y almacenamiento.

La arquitectura Harvard consiste en tener dos memorias: una memoria es para los datos y otra es para las instrucciones.

Requerimientos

Los requerimientos para que el microprocesador opere de forma adecuada es una señal de entrada de 8 bits con la cual el microprocesador podrá llevar una serie de instrucciones dentro del micro; una señal de entrada de 9 bits, la cual tendrá las instrucciones que realizará el microprocesador. Otras entradas que son necesarias son las entradas de reloj y reset, que permiten inicializar el microprocesador.

Internamente el microprocesador debe tener un decodificador que decodifica la entrada que contiene la instrucción, registros que permitan el almacenamiento de datos, una ALU (Unidad lógica aritmética) que hará todas las operaciones del microprocesador, un módulo que controle las salidas del microprocesador, entre otros módulos que puedan ser necesarios.

Se necesitan cuatro salidas del microprocesador tres de ellas son de 8 bits, la primera de ellas es la que contiene la dirección de la siguiente instrucción a realizar, la segunda contiene el dato que se va almacenar en la memoria, otra contiene la dirección de memoria en la que se va a trabajar, y por ultima una salida de un bit que me indica si se va a leer o guardar en la memoria.

Todas estas salidas del microprocesador van a memorias, la memoria ROM que es la memoria de instrucciones, y la memoria RAM que es la memoria de datos; ambas

memorias son asíncronas (no necesitan señal de reloj o rst), con el tamaño suficiente para los 8 bits, en este caso deben tener 256 líneas de dirección.

Todos estos requisitos son necesarios para el correcto funcionamiento del microprocesador.

En la Tabla 1, se encuentra el set de instrucciones del microprocesador, esta tabla contiene las instrucciones y los argumentos que se van a utilizar, así como un breve comentario sobre la instrucción a realizar.

Instruction	Arguments	Description	Comments
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	Stores data at Register RY in [RX] memory address	RY and RX are 3 bits [0,7]
MOVE	RX,RY	Move data form 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 del Microprocesador.

Arquitectura

Microprocesador

El microprocesador es el encargado de la interacción de todas las señales de entrada y salida, así como la interacción entre los módulos internos. Como se puede ver en la imagen 1, es la caja blanca del microprocesador, en ella se puede ver los módulos necesarios en el micro, así como las señales que interactúan entre ellos .

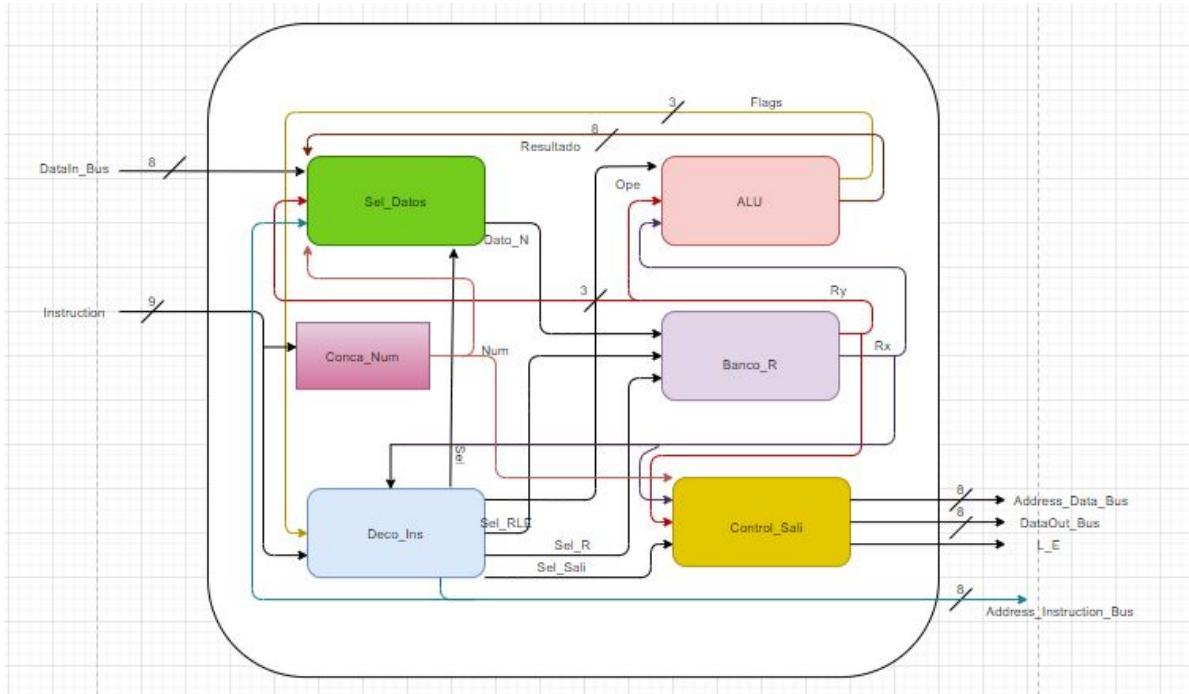


Imagen 1. Caja blanca de microprocesador.

En la Tabla 2 se encuentran las entradas y salidas del sistema, los tamaños de éstas y para qué son necesarias.

Señal	Dirección	Tamaño	Descripción
DataIn_Bus	Entrada	8	Dato de entrada que irá directamente al selector de datos
Instruction	Entrada	9	Entrada que contiene la instrucción codificada
Address_Data_Bus	Salida	8	Indica la dirección de memoria que se necesita para realizar una instrucción
DataOut_Bus	Salida	8	Dato de salida para modificar una dirección de memoria
L_E	Salida	1	Indica si se va a leer o escribir un dato en memoria
Address_Instruction_Bus	Salida	8	Dato de salida que contiene la dirección de la siguiente instrucción a realizar

Tabla 2. Tabla de señales del microprocesador

En la tabla 3 se muestra los sub módulos que componen el microprocesador así como y una breve explicación de su función.

Submodulo	Descripción
Sel_Datos	Este submodulo se encarga de seleccionar el dato que será almacenado en el banco de registros
Conca_Num	Se encarga de concatenar la señal Instruction para obtener en la salida el valor del dato directo correspondiente a un número
Deco_Ins	Decodifica la señal Instruction para obtener todas las señales internas con las que se van a relaizar todas las instrucciones del microprocesador
ALU	Submodulo que va a realizar todas las operaciones matemáticas del microprocesador
Banco_R	Submodulo que va a contener todos los registros del microprocesador
Control_Sali	Submodulo que se encargara del control de salida.

Tabla 3. Tabla de módulos del microprocesador

ALU

El submódulo ALU es el encargado de hacer todas las operaciones aritméticas del microprocesado: suma, resta, corrimiento a la izquierda y derecha, etc. En la imagen 2, se encuentra la caja negra de la ALU, se puede observar las entradas y salidas que son necesarias para su funcionamiento.

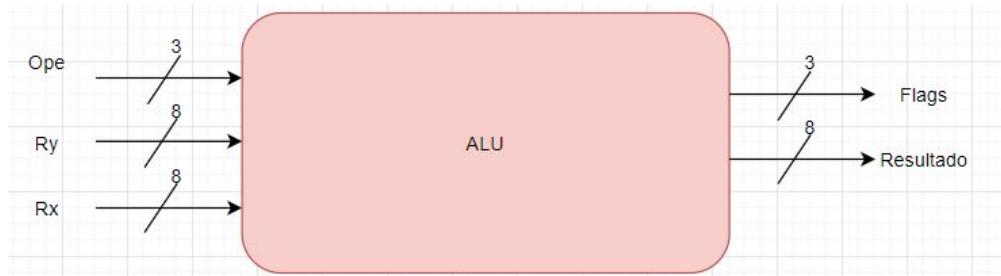


Imagen 2. Caja negra de la ALU

En la tabla 4 se encuentran las entradas y salidas de la ALU, con su tamaño de bits y una breve explicación de lo que es cada señal.

Datos	Direcciones	Ancho (Bits)	Descripción
Ope	Entrada	3	Es la señal instruction decodificada la cual nos dirá cual operación se realizará.
Ry	Entrada	8	Señal que contiene un dato que se va a utilizar para realizar una operación
Rx	Entrada	8	Señal que contiene un dato que se va a utilizar para realizar una operación
Flags	Salida	3	Cada bit de esta señal de salida representa una condición en el resultado de la operación. Z: bit menos significativo, cuando el resultado de la operación es 0. C: Segundo bit menos significativo, cuando hay un acarreo en el resultado. N: bit más significativo, el resultado de la operación es negativo
Resultado	Salida	8	Dato de salida que es el resultado de la operación realizada

Tabla 4. Tabla de señales de la ALU

La tabla 5 (tabla de verdad), tiene el valor de la entrada Ope a la ALU, depende del valor de ésta deberá hacerse una operación. Como se puede observar en la tabla, cada valor de Ope (entrada de ALU) está asignado a una operación.

Ope	Operación
000	Suma: $R0=R0+Rx$
001	Resta: $R0=R0-Rx$
010	Corrimiento: $R0=R0<<Rx$
011	Corrimiento: $R0=R0>>RY$
100	Negado: $R0=\sim Rx$
101	And: $R0=R0\&Rx$
110	Or: $R0=R0 Rx$
111	Xor: $R0=R0^Rx$

Tabla 5. Tabla de verdad de la ALU

En la tabla de Flags (tabla 6), se muestra el bit de la salida Flags, junto a los posibles valores de cada bit, y depende del valor de cada bit, se puede decir si una bandera es verdadera o no, por ejemplo el bit 0 corresponde al Cero, esto quiere decir que si el bit

de la posición cero vale 1 el resultado en la ALU es cero, y si vale 0 el resultado es diferente a cero.

Bit	Valor	Interpretación
0	0	El resultado de la operación no es 0
0	1	El resultado de la operación es 0
1	0	El resultado de la operación no es negativo
1	1	El resultado de la operación es negativo
2	0	El resultado de la operación no lleva un acarreo
2	1	El resultado de la operación lleva un acarreo

Tabla 6. Tabla de Flags

Sel_Datos

Este submódulo se encarga de seleccionar el dato con el cual va a trabajar los registros, depende del valor de Sel es que se va a seleccionar el dato que se mostrará en la salida Data_N. En la imagen 3 se muestra la caja negra de Sel_Datos, en ella se puede observar los datos de entrada y solo tenemos una salida que será el dato seleccionado.

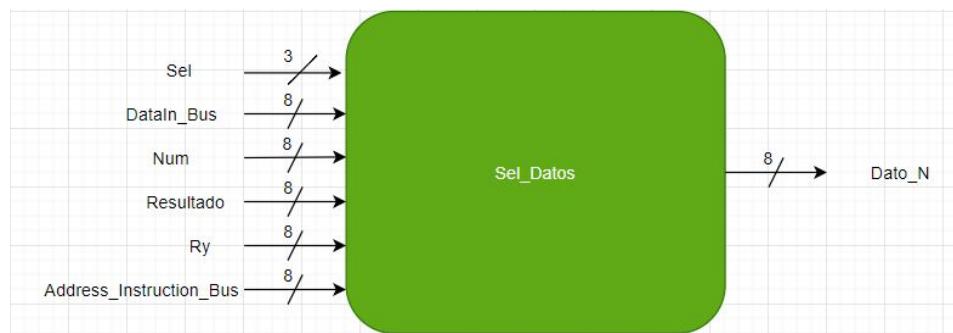


Imagen 3. Caja Negra de Sel_Datos

En la Tabla 7 se encuentran las entradas y salidas de Sel_Datos, así como una breve descripción de éstas.

Señales	Direccion	Ancho (Bits)	Descripcion
DataIn_Bus	Entrada	8	Señal de entrada del Bus de Datos
Sel	Entrada	3	Señal que nos indicara cual dato de entrada será seleccionado
Resultado	Entrada	8	Resultado de la operación realizada en la ALU
Ry	Entrada	8	Entrada que viene del registro RY
Address_Instruction_Bus	Entrada	8	Señal de bus que muestra la dirección de las instrucciones utilizadas.
Num	Entrada	8	Es el valor de un dato directo para las instrucciones que requieran un valor directo
Dato_N	Salida	8	Salida del selector la cual será guardada en el banco de registros

Tabla 7. Tabla de entradas y salidas de Sel_Datos

En la tabla de verdad que se muestra a continuación (Tabla 8), se observa que depende del valor de la entrada Sel, obtendremos determinada señal en la salida, por ejemplo cuando Sel tiene como valor 000 la salida Dato_N será igual a la entrada Resultado, y si Sel es 001 en la salida Dato_N estará el valor de DataIn_Bus.

Sel	Dato_N
000	Resultado
001	DataIn_Bus
010	Num
011	Address_Instruction_Bus
100	Ry
Others	0's

Tabla 8. Tabla de verdad de Sel_Datos

Conca_Num

Este submódulo se encarga de tomar los 3 bits menos significativos de la entrada Instruction, para obtener un nuevo valor del dato directo correspondiente a un número. En la Imagen 3 se encuentra la tabla negra de Conca_Num en ella se observa la entrada Instruction y la única salida Num, que es el numero dato/número.

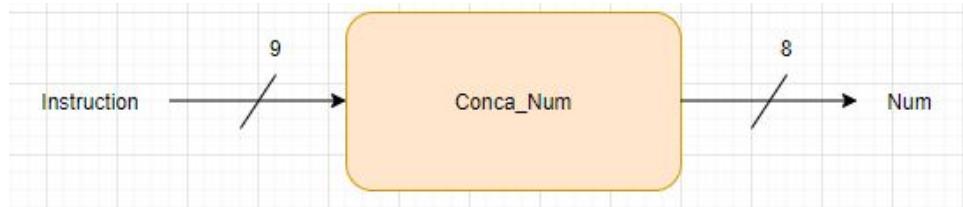


Imagen 4. Caja negra de Conca_Num

En la tabla 9 se encuentran las entradas y salidas de Conca_Num, su tamaño en bits y una breve explicación de cada señal.

Señal	Dirección	Ancho(bits)	Descripción.
Instruction	Entrada	9	Señal de entrada que contiene la instrucción a realizar.
Num	Salida	8	Señal de dato directo que corresponde a un numero para las instrucciones que requieran un valor directo.

Tabla 9. Tabla de entradas y salidas de Conca_Num

En la tabla 10 se encuentra la forma o la manera en la que se concatenó el número, los bits más significativos de la salida Num tendrán del valor de cero, pero los tres bits menos significativos serán igual a los tres bits menos significativos de la entrada Instruction.

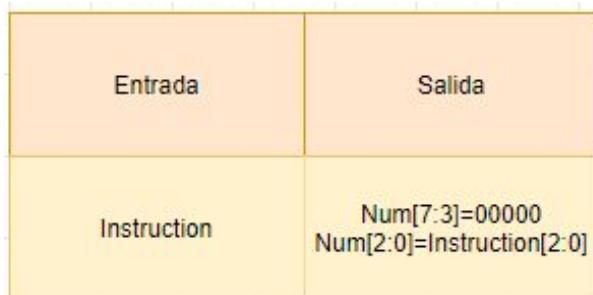


Tabla 10. Tabla de concatenación

Deco_Ins

Este submódulo es el más importante de todos, ya que este se encarga de la decodificación de la instrucción y además es el encargado de mandar las señales a los otros submódulos para que se haga la instrucción correspondiente.

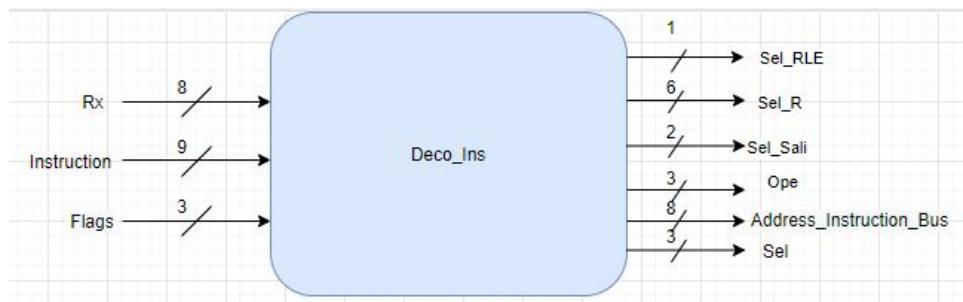


Imagen 5. Caja negra de Deco_Ins

En la Tabla 11 se encuentran las entradas y salidas del Deco_Ins, su tamaño en bits y una breve explicación de la señal.

Datos	Direcciones	Ancho(Bits)	Descripción
Instruction	Entrada	9	Señal de entrada que contiene la instrucción codificada con el OP Code y los operandos dependiendo de la nueva instrucción.
Flags	Entrada	3	Indica la bandera que generó la ALU en su resultado. Saltos condicionales y no condicionales
Rx	Entrada	8	Es un dato que viene del registro, el cual contiene una dirección para modificar el PC.
Sel_RLE	Salida	1	Señal encargada de leer o escribir en el banco de registros
Sel_R	Salida	6	Esta señal selecciona los registros Ry y Rx dependiendo de la instrucción que se vaya a ejecutar.
Sel_Sali	Salida	2	Señal que selecciona la operación que hará el control de salida
Ope	Salida	3	Es la señal instrucción decodificada la cual nos dirá cuál operación se realizará la ALU.
Address_Instruction_Bus	Salida	8	Señal de bus que contiene la dirección de la instrucción que se va a ejecutar (PC).
Sel	Salida	3	Señal que selecciona qué dato se va a guardar en el banco de registros.

Tabla 11. Entradas y salidas de Deco_Ins

La tabla de verdad (Tabla 12) del Deco_Ins, tiene todos los valores de las salidas, estas dependen de la entrada Instruction, depende de los 3 bits más significativos para que realice una instrucción; los 6 bits menos significativos de la entrada Instruction tienen los argumentos con los que se va a trabajar, esto quiere decir que tiene los registros con los que se va a trabajar, en dado caso que se vaya a realizar una operación tiene el valor de Ope, la condición para el JUMP.

Instrucción	Argumentos	Instruction [8:0] [5:3] [2:0]	Ope	Sel_R [5:3] [2:0]	Sel_RLE	Sel_Sali	Sel	Address_Instruction_Bus
LOAD	Rx,#Num	001,Rx,Num	0	000,Rx	1	00	010	PC+1
LOAD	Rx,[Ry]	010,Rx,Ry	0	Ry,Rx	1	01	001	PC+1
STORE	#Num	011,Rx,Num	0	000,Rx	0	10	000	PC+1
STORE	[Rx],Ry	100,Rx,Ry	0	Ry,Rx	0	11	000	PC+1
MOVE	Rx,Ry	101,,Rx,Ry	0	Ry,Rx	1	00	100	PC+1
MATH	Rx,OP	110,Rx,Op	OP	R0,Rx	1	00	000	PC+1
JUMP	Rx,COND	111,Rx,COND	0	COND=1 -> a)000,R7 b)000,Rx else -> 000,Rx	if COND=0 -> a)1 b)0 else -> 1	00	if Cond=1-->001 others->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 12. Tabla de verdad de Deco_Ins

Este submódulo también es el encargado de hacer el JUMP (salto) en el PC, este salto depende de la entrada Flags, que depende de si ésta es verdadera o no se realiza un JUMP y cada que se ejecuta una instrucción también se hace un JUMP en el PC.

Banco_R

El Banco_R, es un banco de registros, este banco contiene 8 registros que se pueden utilizar para almacenar datos dentro del microprocesador. En la Imagen 6 se encuentra la caja negra del Banco_R en ella se puede observar las entradas y salidas de este submódulo..

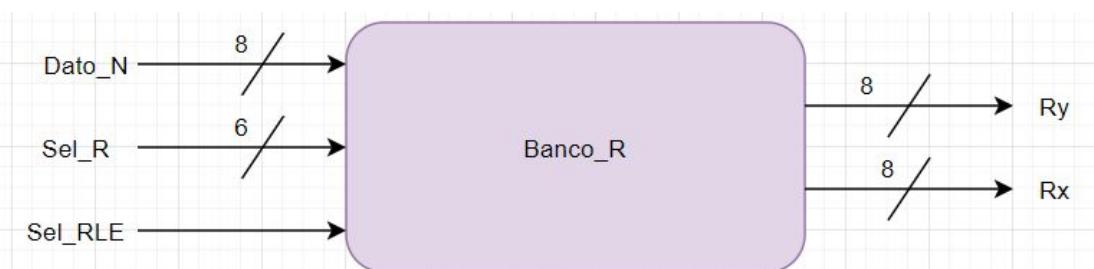


Imagen 6.Caja negra de Banco_R

En la Tabla 13 se encuentran las entradas y salidas del submódulo, con su tamaño correspondiente en bits y una breve explicación de estas señales.

Datos	Direcciones	Ancho (Bits)	Descripción
Dato_N	Entrada	8	Dato que proviene del selector de datos y que será almacenado en el banco de registros
Sel_R	Entrada	6	Esta señal de entrada selecciona el registro en el cual se va a hacer la instrucción
Sel_RLE	Entrada	1	Señal que permite leer o almacenar datos en los registros.
Ry	Salida	8	Señal que representa uno de los datos con los que se va a realizar una instrucción.
Rx	Salida	8	Señal que representa uno de los datos con los que se va a realizar una instrucción.

Tabla 13. Entradas y salidas de Banco_R

En la siguiente tabla se puede observar que la entrada Sel_RLE dependiendo de su valor (1 o 0) es que se va a leer o escribir en los registros; y a su vez podemos observar que Rx y Ry ambos son registros que se encuentran dentro de Banco_R se seleccionan mediante la entrada Sel_R. donde los 3 bits menos significativos son para el registro Rx y los 3 bits más significativos para Ry.

Sel_RLE	Rx	Ry
1 (Leer)	R(Sel_R[2:1])	R(Sel_R[5:3])
0 (Escribir)	R(Sel_R[2:1])=Dato_N	R(Sel_R[5:3])

Tabla 14. Asignación de Rx y Ry

Control_Sali

Este submódulo se encarga del control de las salidas que corresponden al dato, tiene como función mostrar en la salida la dirección y el dato que corresponde a la instrucción, así como si éste se va a leer o escribir. En la Imagen 6 se muestra la caja negra de Control_Sali, sus entradas y salidas.



Imagen 6. Caja negra de Control_Sali

En la tabla 15 se muestran las entradas y salidas de Control_Sali, el tamaño en bits de éstas y una breve explicación de cada señal.

Señal	Dirección	Ancho(bits)	Descripción.
Ry	Entrada	8	Dato con el cual se puede realizar una instrucción definida
Rx	Salida	8	Dato con el cual se puede realizar una instrucción definida
Sel_Sali	Entrada	2	Señal que selecciona que va realizar el control de salida
Address_Data_Bus	Salida	8	Dato de salida que indica la dirección de memoria que se necesita para realizar una instrucción
DataOut_Bus	Salida	8	Dato de salida para modificar una dirección de memoria
L_E	Salida	1	Señal de salida que indica si se va a guardar o leer un dato en memoria
Num	Entrada	8	Dato de entrada para la instrucción Store de un número inmediato

Tabla 15. Entradas y salidas de Control_Sali

En la siguiente tabla (Tabla 16) se muestra la instrucción del microprocesador junto al valor que debe tener las señales de salida que dependen de la señal de entrada Sel_Sali.

Instruction	Sel_Sali	DataOut_Bus	Address_Data_Bus	L_E
NOP	00	0's	0's	0
LOAD (dato de la dirección Ry)	01	0's	Ry	0
Store, #NUM en la dirección [Rx]	10	Num	Rx	1
Store, dato Ry en la dirección [Rx]	11	Ry	Rx	1

Tabla 16. Tabla de verdad de Control_Sali

Multiplicación

La multiplicación con el microprocesador se realiza mediante una serie de instrucciones dentro del mismo microprocesador, de tal forma que nos permita observar el resultado en nuestro dato de salida. En la tabla 17 se encuentran las instrucciones para poder realizar la multiplicación.

#	Instruction	Argumentos	Descripción	=
0	Load	R1,Num	R1<-num1	=
1	Load	R2,Num	R2<-Num2	=
2	Load	R3,1	R3<-1	=
3	Load	R4,4	R4<-4	=
4	Move	R0,R5	R0<-R5	=
5	Math	R1,000	R0<-R0+R1	=
6	Move	R5,R0	R5<-R0	=
7	Move	R0,R2	R0<-R2	=
8	Math	R3,001	R0<-R0-R3	=
9	Move	R2,R0	R2<-R0	=
10	Jump	R4,3	if Z=0	=
11	Move	R0,R5	R0=R5	=

The diagram shows a control flow loop starting from instruction 4 and ending at instruction 10. A vertical dashed green line connects the = symbols of instruction 4 and 5. A curved black arrow points from the = symbol of instruction 5 back to the = symbol of instruction 4. To the right of this loop, the text "Z=0" is written.

Tabla 17. Instrucciones para la multiplicación

División

La división al igual que la multiplicación se lleva a cabo mediante una serie de instrucciones dentro del microprocesador, en la tabla 18 se pueden observar estas instrucciones y la secuencia que debe seguir para que se realice bien la operación.

#	Instruction	Argumentos	Descripción	
0	Load	R1,Num	R1<-Num	
1	Load	R2,Num	R2<-Num2	
2	Load	R3,1	R<-1	
3	Load	R4,4	R4<-4	
4	Move	R0,R5	R0<-R5	
5	Math	R3,0	R0<-R3+R0	
6	Move	R5,R0	R5<-R0	
7	Move	R0,R1	R0<-R1	
8	Math	R2,1	R0<-R0-R2	
9	Move	R1,R0	R1<-R0	
10	Jump	R4,7	if N=0 R0<-R5	
11	Move	R0,R5	R0<-R5	
12	Math	R3,1	R0<-R0-R5	

Tabla 18. Instrucciones para la división.

Implementación

En la Imagen 7 se muestra el Microprocesador conectado a la memoria de datos y a la memoria de instrucciones, se observa las conexiones entre ellos para que el microprocesador opere de la forma correcta.

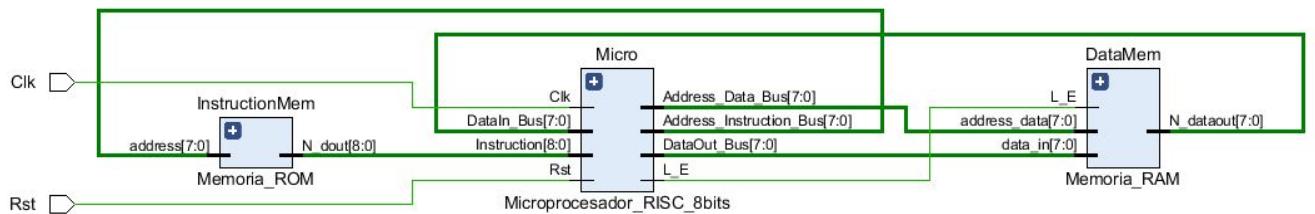


Imagen 7. Esquema del microprocesador

Se observa que la memoria ROM es la memoria de instrucciones y la memoria RAM es la memoria de datos, se puede comprobar que el microprocesador si tiene arquitectura Harvard

Pruebas

MicroMem

La siguiente simulación se observa el correcto funcionamiento del Microprocesador con memorias, depende del valor de la señal Instruction es lo que se va hacer en el microprocesador.

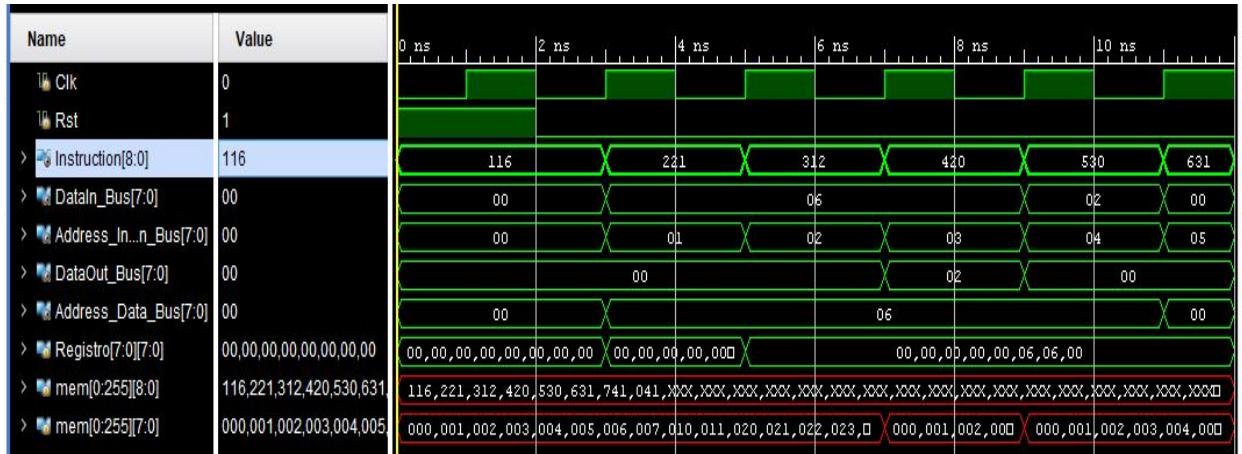


Imagen 8. Simulacion de MicroMem

En el primer ciclo de reloj la señal Instruction en octal vale 116 lo que quiere decir que se va a hacer un load y que el número 6 se va a guardar en el registro 1, y en la señal de registro en el registro uno se guarda el 6. En el segundo ciclo de reloj Instruction en octal tiene un valor de 221 también es un Load y la dirección de R1 se va a guardar en R2, mientras tanto en el Pc se suma uno.

Cada ciclo reloj va cambiando de instrucción, estas instrucciones se encuentran en la memoria de instrucciones (ROM).

Microprocesador_RISC_8bits

En la simulación del microprocesador sin memorias, cada ciclo de reloj hay un cambio en las señales. En la señal de salida Address_Instruction_Bus cada que se ejecuta una instrucción en el pc se suma uno. Cuando hay un Store la señal L_E esta en alto, lo que quiere decir que se va a escribir en memoria. La señal Instruction toma los valores que nosotros le asignamos en el Test bench.

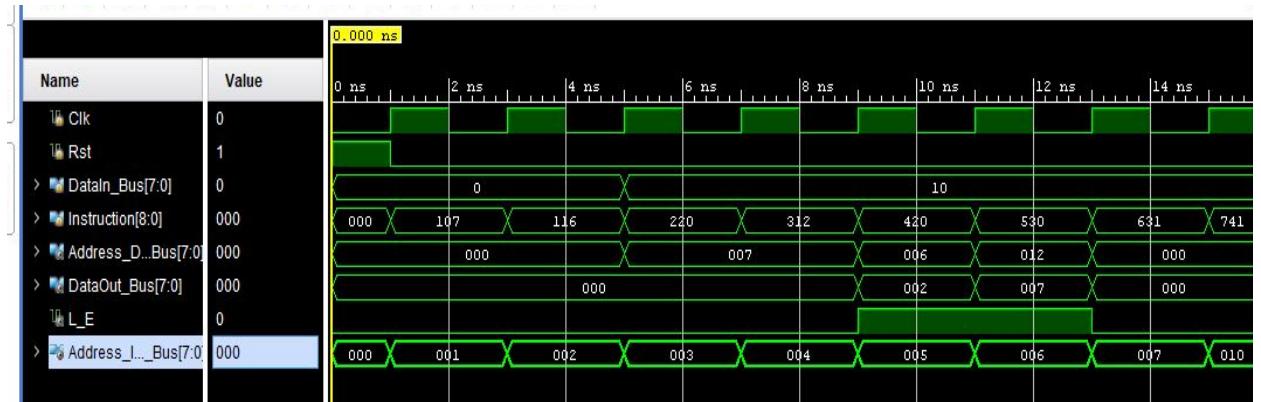


Imagen 9. Simulación de Microprocesador_RISC_8bits

ALU

La simulación de la ALU se demostró de manera sencilla el correcto funcionamiento de este submódulo. Del tiempo 0 al 2 Ope=0 por lo tanto se va hacer una suma entre los dos datos de entradas que valen 3 y 4 respectivamente, por lo tanto en la señal Resultado obtenemos un 7. Del tiempo 2 al 4 Ope=1 por lo tanto se hará una resta entre Rx y Ry que va a ser igual a -1. Esto demuestra el correcto funcionamiento del submódulo y se puede observar en la simulación.

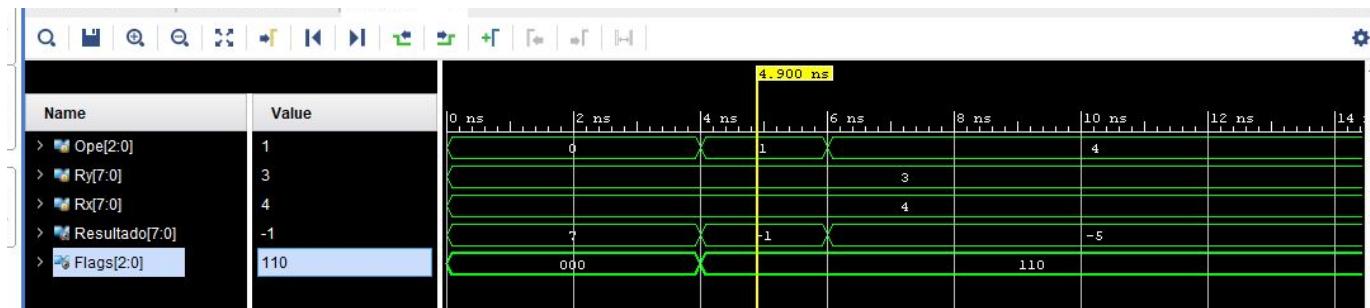


Imagen 10. Simulacion de ALU

Sel_Datos

En la simulación de Sel_Datos la única señal de entrada que cambia de valor es Sel cada 2 tiempos cambia esta señal, cuando Sel vale 0 la señal que se muestra en la salida es REsultado; cuando SEl vale 1 el dato que se muestra en la salida DataIn-Bus, cuando Sel vale 2 la señal de salida vale Num. Esto comprueba el correcto funcionamiento del submódulo.

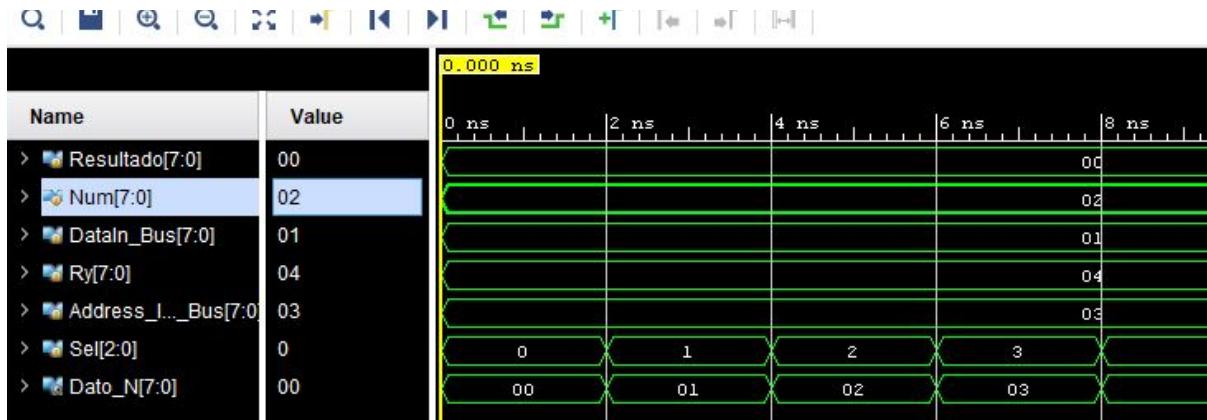


Imagen 11. Simulación de Sel_Datos

Conca_Num

En la simulación de Conca_Num se puede observar que cada que cambia la señal de entrada Instruction cambia el valor de Num, tomando los tres bits menos significativos de la señal de entrada, esto quiere decir que funciona correctamente el módulo.

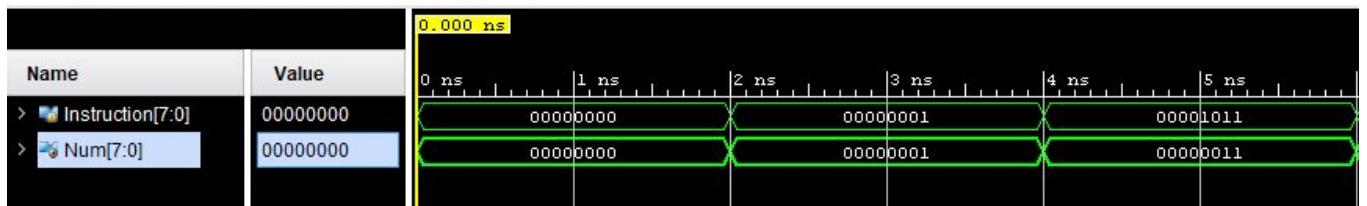


Imagen 12. Simulación de Conca_Num

Deco_Ins

La simulación de Deco_Ins, depende del valor de la señal de entrada Instruction se obtendrán las valores para la salida del sistema, por ejemplo cuando la Instruction en octal vale 100 en la salida Sel se obtendrá un 2 y Ope vale cero, y su vez Sel_R vale 00 y Sel_Sali, todos los valores de salida son los correspondientes a la tabla de verdad que se mostró en arquitectura.

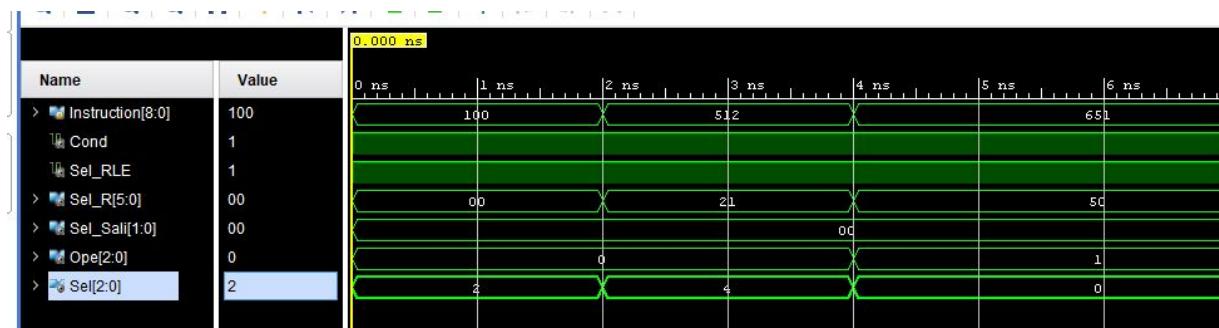


Imagen 13. Simulación Deco_Ins

Banco_R

La simulación del banco de registros (Banco_R) cada ciclo del reloj va cambiando las señales de entrada, y según lo que vale Sel_R son los registros en los que se va a trabajar se puede observar que del tiempo 10 al 12 Sel_R en octal vale 03 esto quiere decir que el que se va a trabajar con el registro 0 y el registro 3 (Ry, Rx respectivamente) en ese instante Dato_N vale 0 y se almacena en el Rx, todo esto depende de la decodificación de Instruction.

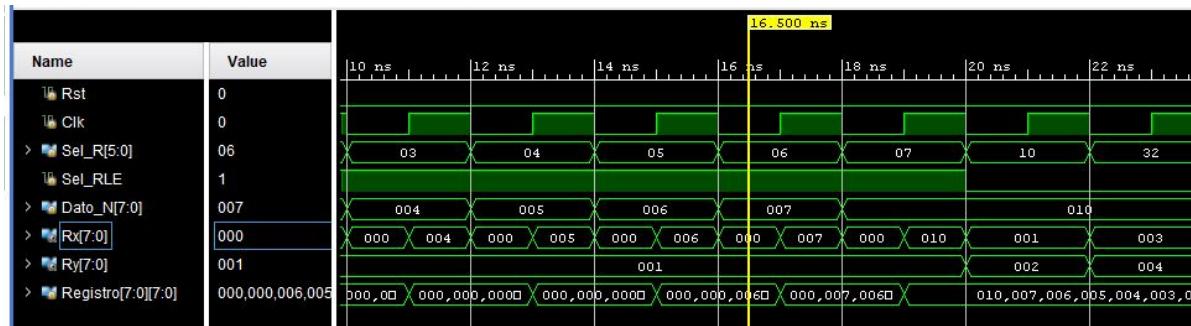


Imagen 14. Simulación de Banco_R

Control_Sali

En el control de salida cada 2 tiempos Sel_Sali está cambiando por lo tanto las señales de salida también. Cuando Sel_Sali vale 1 DataOut_Bus es cero y Address_Data_Bus toma el valor de Ry ya que es el segundo Load en nuestro set de instrucciones. Después Sel_Sali vale 3 y es el primer store por lo tanto L_E vale 1 y DataOut_Bus vale Ry y Address_Data_Bus vale Rx.



Imagen 15. Simulación de Control_Sali

Multiplicación

En la Imagen 16 se puede observar la simulación de la multiplicación con el microprocesador, esta multiplicación se llevó a cabo mediante una serie de instrucciones. Los números que se están multiplicando son el 4 y el 2. En la señal de Registros se puede observar que en registro cero (R0) se encuentra el resultado de la multiplicación.

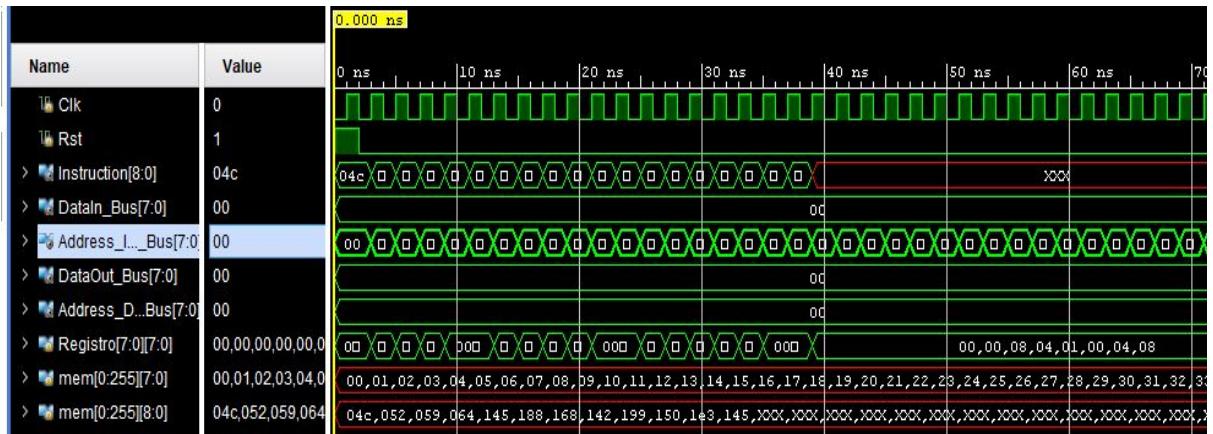


Imagen 16. Simulación de la multiplicación

División

Al igual que en la multiplicación, para poder llevar a cabo la multiplicación es necesario llevar a cabo una serie de instrucciones que nos permitan la división de dos números, que en este caso son el 4 y el 2, el resultado de esta división es dos, y el resultado se muestra en el registro R0.

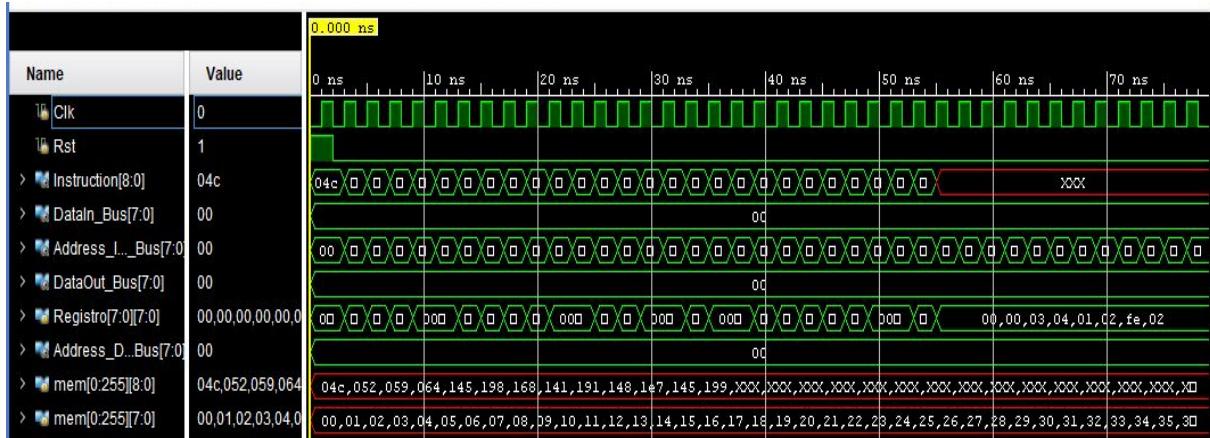


Imagen 17. Simulación de división

Análisis de resultado

Es necesario saber la capacidad de un FPGA, debido a que el microprocesador requiere una mínima cantidad de LUTs y registros para poder operar, todo esto se analiza y se muestra en la Tabla 19.

Como se observa en la tabla se tiene un total de 20800 LUTs de las cuales el microprocesador sólo utiliza 221, de los 41600 registros (Slice Registers) el microprocesador sólo utiliza 108.

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Bonded IOB (106)	BUFGCTRL (32)
Microprocesador_RISC_8...	221	108	24	44	1
Instrucciones (Deco_I...)	40	11	0	0	0
Operaciones (ALU)	5	0	0	0	0
Registros (Banco_R)	175	64	24	0	0
Salida (Control_Sali)	1	33	0	0	0

Tabla 19. Microprocesador en el FPGA

En la siguiente imagen se observa en porcentaje cuánto es que ocupa el microprocesador, siendo IO el mayor de todos con un 42% (exactamente 41.51%)

Resource	Utilization	Available	Utilization %
LUT	221	20800	1.06
FF	108	41600	0.26
IO	44	106	41.51

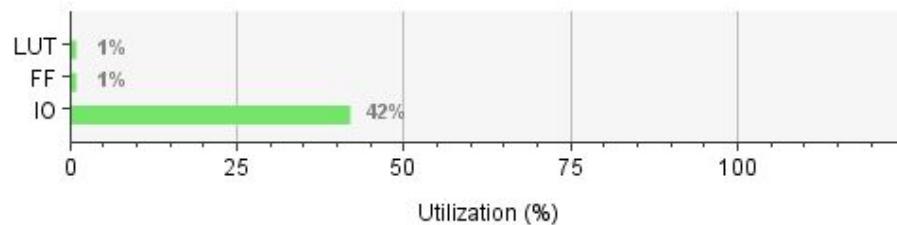


Imagen 18. Porcentaje del Microprocesador en el FPGA.

Conclusiones

Se obtuvo el correcto funcionamiento del microprocesador, se cumplió con lo estipulado en el set de instrucciones que nos proporcionó el profesor de esta materia. Para llegar a este punto hubo un largo proceso y un gran trabajo en equipo, pues se tenía que cumplir con algo estipulado, y teníamos que hacer cajas negras, blancas y códigos que nos permitieran eso.

Se tuvieron varios errores al momento de simular, pues algunos códigos fueron a base de prueba y error, y otros códigos salieron a la primera.

Pero este proyecto nos mostró la complejidad de un microprocesador, también se vio que hay distintas maneras de un microprocesador, pero sobre todo nos mostró que nosotros podemos desarrollar nuestros propios microprocesadores, con nuestro set de instrucciones con la arquitectura que deseemos.

Referencias

[1]Samir Palnitkar (2003, February 21), “Verilog HDL. A Guide to Digital Design and

Synthesis” Second Edition.

[2]A.P. Godse (2009) “Microprocessors and Interfacing” First Edition

Apéndices

Apéndice A. Código de implementación

MicroMem

```

1. `timescale 1ns / 1ps
2.///////////
3.// Company:
4.// Engineer:
5.//
6.// Create Date: 11.11.2020 10:06:47
7.// Design Name:
8.// Module Name: tb_MicroMem
9.// Project Name:
10.// Target Devices:
11.// Tool Versions:
12.// Description:
13.//
14.// Dependencies:
15.//
16.// Revisión:
17.// Revision 0.01 - File Created
18.// Additional Comments:
19.//
20.///////////
21.
22.
23.module tb_MicroMem;
24.
25.reg Clk;
26.reg Rst;
27.
28.MicroMem uut(
29. .Clk(Clk),
30. .Rst(Rst)
31. )
32.
33.initial
34.begin
35.Rst=1;
36.Clk=0;
37.
38.#2 Rst=0;
39.#10 $finish;

```

```

40.end
41.
42.always
43. #1 Clk = !Clk;
44.endmodule
45.

```

Microprocesador_8bits

```

1. `timescale 1ns / 1ps
2. ///////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 10.11.2020 19:44:53
7. // Design Name:
8. // Module Name: Microprocesador_RISC_8bits
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. ///////////////////////////////////////////////////
21.

22.
23.

24.module Microprocesador_RISC_8bits(
25.     input      Clk,
26.     input      Rst,
27.     input [8:0] Instruction,
28.     input [7:0] DataIn_Bus,
29.     output [7:0] Address_Instruction_Bus,
30.     output [7:0] DataOut_Bus,
31.     output [7:0] Address_Data_Bus,

```

```

32.output    L_E
33. );
34.//////////35
.///////////
36.wire [7:0]W_Num;
37.wire[2:0] W_Sel;
38.wire [5:0] W_Sel_R;
39.wire W_Sel_RLE;
40.wire [7:0] W_Dato_N;
41.wire [7:0] W_RX;
42.wire [7:0] W_RY;
43.wire [2:0] W_Ope;
44.wire [7:0] W_Resultado;
45.wire [2:0] W_Flags;
46.wire [1:0] W_Sel_Sali;
47.//////////48
./////////
49.Sel_Datos Selector (
50. .DataIn_Bus(DataIn_Bus),
51 .Resultado(W_Resultado),
52. .Ry(W_RY),
53. .Address_Instruction_Bus(Address_Instruction_Bus),
54 .Num(W_Num),
55. .Sel(W_Sel),
56. .Dato_N(W_Dato_N)
57.);
58.
59.//////////60
.///////////
61.ALU operaciones (
62. .Ope(W_Ope),
63. .Ry(W_RX),

```

```

64. .Rx(W_RY),
65. .Resultado(W_Resultado),
66. .Flags(W_Flags)
67.);

68.//////////69
.//////////

70.Conca_Num Inmediato (
71 .Instruction(Instruction),
72. .Num(W_Num)
73.);

74.//////////75
.//////////

76.Banco_R Registros (
77. .Sel_R(W_Sel_R), // Tres bits para cada uno,Sel_R[2:0] para Rx y Sel_R[5:3] para Ry
78. .Dato_N(W_Dato_N),// Dato de entrada que sera almacenado
79. .Rx(W_RX),
80. .Ry(W_RY),
81. .Sel_RLE(W_Sel_RLE), // Con 0 va a leer y con 1 a escribir
82. .Rst(Rst),
83. .Clk(Clk)
84.);

85.//////////86
.//////////

87.Deco_Ins Instrucciones (
88. .Rx(W_RY),
89. .Instruction(Instruction),
90. .Flags(W_Flags),
91. .Sel_RLE(W_Sel_RLE), //Salida que dice si se va a leer o guardar
92. .Sel_R(W_Sel_R),// Selecciona el registro a utilizarse
93 .Sel_Sali(W_Sel_Sali), //Señal que controla la salida
94. .Ope(W_Ope), //Operación de la ALU
95. .Address_Instruction_Bus(Address_Instruction_Bus),

```

```

96. .Sel(W_Sel), //Sel de datos
97. .Clk(Clk),
98. .Rst(Rst)
99.);

100.//////////10
1.//////////

102.Control_Sali Salida (
103. .Ry(W_RY),
104. .Rx(W_RX),
105 .Num(W_Num),
106. .Sel_Sali(W_Sel_Sali),
107. .Address_Data_Bus(Address_Data_Bus),
108. .DataOut_Bus(DataOut_Bus),
109. .L_E(L_E),
110. .Clk(Clk),
111. .Rst(Rst)
112.);

113.endmodule

```

ALU

```

1. `timescale 1ns / 1ps
2. //////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 10.11.2020 19:30:55
7. // Design Name:
8. // Module Name: ALU
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:

```

```

19. //
20. //////////////////////////////////////////////////////////////////
21.
22.
23.module ALU(
24.input [2:0] Ope,
25. input [7:0] Ry,
26. input [7:0] Rx,
27. output [7:0] Resultado,
28. output [2:0] Flags
29. );
30. reg [8:0] R0;
31.
32. always@*
33. begin
34. case (Ope)
35. 3'b000: begin R0<=Ry+Rx;end
36. 3'b001: begin R0<=Ry-Rx; end
37. 3'b010: begin R0<=Ry<<Rx; end
38. 3'b011: begin R0<=Ry>>Rx;end
39. 3'b100: begin R0<=~Rx; end
40. 3'b101: begin R0<=Ry&Rx;end
41. 3'b110: begin R0<=Ry|Rx;end
42. 3'b111: begin R0<=Ry^Rx;end
43. endcase
44. end
45. assign Resultado= R0[7:0]; //Los 7 bits menos significativos son el resultado de la operacion
46. //Bit más significativo es el de acarreo
47. //Banderas
48. assign Flags[0]=&(~R0); //Bit cero
49. assign Flags[1]= R0[8]; //Bit de acarreo C

```

```

50. assign Flags[2]=R0[7]; //N:Negativo
51.
52.
53.
54.endmodule

```

Sel_Datos

```

1. timescale 1ns / 1ps
2./////////////////////////////////////////////////////////////////////////
3.// Company: Universidad Autonoma de Zacatecas
4.// Engineer: Roboticos
5.//
6.// Create Date: 18.10.2020 15:48:20
7.// Design Name: Sel_Datos
8.// Module Name: Sel_Datos
9.// Project Name: Microprocesador_RISC_8bits
10.// Target Devices:
11.// Tool Versions:
12.// Description:
13.//
14.// Dependencies:
15.//
16.// Revision:
17.// Revision 0.01 - File Created
18.// Additional Comments:
19.//
20./////////////////////////////////////////////////////////////////////////
21.
22.
23.module Sel_Datos(
24.    input [7:0] DataIn_Bus,
25.    input [7:0] Resultado,
26.    input [7:0] Ry,
27.    input [7:0] Address_Instruction_Bus,
28.    input [7:0] Num,
29.    input [2:0] Sel,
30.    output reg [7:0] Dato_N
31. );
32.
33.
34.
35.    always@*
36.    begin
36.        case (Sel)

```

```

37. 3'b000 :begin Dato_N<=Resultado;end
38. 3'b001 :begin Dato_N<=DataIn_Bus;end
39. 3'b010 :begin Dato_N<=Num;end
40. 3'b011 :begin Dato_N<=Address_Instruction_Bus;end
41. 3'b100 :begin Dato_N<=Ry;end
42. default: begin Dato_N<=0;end
43. endcase
44.
45. end
46.
47.endmodule

```

Conca_Num

```

1. `timescale 1ns / 1ps
2.///////////
3.// Company:
4.// Engineer:
5.//
6.// Create Date: 10.11.2020 23:00:35
7.// Design Name:
8.// Module Name: Conca_Num
9.// Project Name:
10.// Target Devices:
11.// Tool Versions:
12.// Description:
13.//
14.// Dependencies:
15.//
16.// Revision:
17.// Revision 0.01 - File Created
18.// Additional Comments:
19.//
20.///////////
21.
22.
23.module Conca_Num
24.#(parameter n = 8 )
25.(
26.input      [n:0]  Instruction,
27.output reg [n-1:0] Num
28.);
29.reg [n-1:0]VD;
30. always@*
31.begin
32.

```

```

33.VD = 8'b0
34.Num <= {VD[7:3],Instruction[2:0]};
35.
36.end
37.endmodule

```

Deco_Ins

```

1.timescale 1ns / 1ps
2.///////////
3.// Company:
4.// Engineer:
5.//
6.// Create Date: 10.11.2020 19:26:28
7.// Design Name:
8.// Module Name: Deco_Ins
9.// Project Name:
10.// Target Devices:
11.// Tool Versions:
12.// Description:
13.//
14.// Dependencies:
15.//
16.// Revision:
17.// Revision 0.01 - File Created
18.// Additional Comments:
19.//
20.///////////
21.
22.
23.module Deco_Ins(
24.input [7:0] Rx, //Valor que viene del banco de registros
25.input [8:0] Instruction,
26.input [2:0] Flags,
27.output Sel_RLE, //Salida que dice si se va a leer o guardar
28.output [5:0] Sel_R, //Selecciona el registro a utilizarse
29.output [1:0] Sel_Sali, //Señal que controla la salida
30.output [2:0] Ope, //Operación de la ALU
31.output [7:0] Address_Instruction_Bus,
32.output [2:0] Sel, //Sel de datos
33.input Clk,
34.input Rst
35.);
36.
37.wire [3:0]CondJ; //Cable que une al Jum y al Decodificador
38.reg [2:0] NFlags; //Registro para almacenar las banderas despues de una math

```

```

39.
40.Decodificador c_Decodificador(
41..Instruction(Instruction),
42..Cond(CondJ),
43..Sel_RLE(Sel_RLE),
44..Sel_R(Sel_R),
45..Sel_Sali(Sel_Sali),
46..Ope(Ope),
47..Sel(Sel)
48.
49.
50.Jump c_Jump(
51..Flags(NFlags),
52..Rx(Rx),
53..Cond(CondJ),
54..Clk(Clk),
55..Rst(Rst),
56..Address_Instruction_Bus(Address_Instruction_Bus)
57.);
58.
59.
60.always @(posedge Clk, posedge Rst) begin
61.if(Rst)
62.begin
63.NFlags<=0;
64.end
65.else
66.if (Instruction[8:6]==3'b110)
67.NFlags<=Flags;
68.    end
69.endmodule

```

Decodificador

1. `timescale 1ns / 1ps
2. /////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 10.11.2020 19:27:56
7. // Design Name:
8. // Module Name: Decodificador
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //

```

14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. /////////////////////////////////
21.
22.
23. module Decodificador(
24.     input [8:0] Instruction,
25.     output reg [3:0] Cond,
26.     output reg Sel_RLE, //Señal que indica si se va a leer o escribir
27.     output reg [5:0] Sel_R, //Señal para seleccionar los registros
28.     output reg [1:0] Sel_Sali, //Señal que controla el modulo Control_Sali
29.     output reg [2:0] Ope, //Señal que indica que operación
30.     output reg [2:0] Sel //Selecciona el dato en el modulo Sel_Dato
31. );
32.
33.     always @(Instruction) begin
34.         case (Instruction [8:6])
35.             //Load Rx,#Num
36.             3'b001: begin
37.                 Ope<=0; Sel_R<= {3'b000,Instruction[5:3]}; Sel_RLE<=1;
38.                 Sel_Sali<=2'b00; Sel<=3'b010; Cond<=4'b0001; end
39.             //Load Rx,[Ry]
40.             3'b010: begin
41.                 Ope<=0; Sel_R<={Instruction[2:0],Instruction[5:3]}; Sel_RLE<=1;
42.                 Sel_Sali<=2'b01; Sel<= 3'b001;Cond<=4'b0001; end
43.             //Store #Num
44.             3'b011: begin
45.                 Ope<=0; Sel_R<= {3'b000, Instruction[5:3]}; Sel_RLE<=0;
46.                 Sel_Sali<=2'b10; Sel<=3'b010; Cond<=4'b0001; end
47.             //Store [Rx],Ry
48.             3'b100: begin
49.                 Ope<=0; Sel_R<={Instruction[2:0], Instruction[5:3]}; Sel_RLE<=0;
50.                 Sel_Sali<=2'b11; Sel<=3'b100;Cond<=4'b0001; end
51.             //Move Rx,Ry
52.             3'b101: begin
53.                 Ope<=0; Sel_R<={Instruction[2:0], Instruction[5:3]}; Sel_RLE<=1;
54.                 Sel_Sali<=2'b00; Sel<=3'b100;Cond<=4'b0001; end
55.             //Math
56.             3'b110: begin
57.                 Ope<=Instruction[2:0]; Sel_R<={Instruction[5:3],3'b000}; Sel_RLE<=1;
58.                 Sel_Sali<=2'b00; Sel<=3'b000;Cond<=4'b0001; end
59.             //Jump
60.             3'b111: begin
61.                 if (Instruction[2:0]==3'b001) // jump sin condicion y guardar pc en R7

```

```

56.           begin
57.             Ope<=0; Sel_R<={Instruction[5:3], 3'b111}; Sel_RLE<=1;
      Sel_Sali<=2'b00; Sel<=3'b011; Cond<={1'b1, Instruction[2:0]}; end
58.
59.           else
60.             begin Ope<=0; Sel_R<={Instruction[5:3], 3'b000}; Sel_RLE<=0;
      Sel_Sali<=2'b00; Sel<=3'b011; Cond<={1'b1, Instruction[2:0]}; end
61.           end
62.           //Nop
63.           3'b000: begin
64.             Ope<=0; Sel_R<=0; Sel_RLE<=0; Sel_Sali<=2'b00;
      Sel<=3'b111; Cond<=4'b0001; end
65.
66.           default: begin Ope<=0; Sel_R<=0; Sel_RLE<=0; Sel_Sali<=2'b00;
      Sel<=3'b111; Cond<=4'b0001; end
67.
68.           endcase
69.
70.
71.
72.       end
73.
74. endmodule

```

Jump

```

21.
22.
23. module Jump(
24.   input [2:0] Flags,
25.   input [7:0] Rx,
26.   input [3:0] Cond,
27.   output [7:0] Address_Instruction_Bus,
28.   input Clk,
29.   input Rst
30. );
31.
32. reg [7:0]pc; //Guarda el Address_Instruction_Bus
33.
34.   always@(posedge Clk,posedge Rst)
35.     begin
36.       if(Rst)
37.         pc<=0;
38.       else
39.         case(Cond)
40.           4'b1000: pc <= Rx;
41.           4'b1001: pc <= Rx;
42.
43.           4'b1010:
44.             begin
45.               if(Flags[0])
46.                 pc <= Rx;
47.               else
48.                 pc = pc+1'b1;
49.             end
50.           4'b1011:
51.             begin
52.               if(~Flags[0])
53.                 pc <= Rx;
54.               else
55.                 pc= pc+1'b1;
56.             end
57.           4'b1100:
58.             begin
59.               if(Flags[1])
60.                 pc <= Rx;
61.               else
62.                 pc = pc+1'b1;
63.             end
64.           4'b1101:
65.             begin
66.               if(~Flags[1])
67.                 pc <= Rx;
68.               else

```

```

69.          pc = pc+1'b1;
70.      end
71.      4'b1110:
72.      begin
73.          if(Flags[2])
74.              pc <= Rx;
75.          else
76.              pc= pc+1'b1;
77.          end
78.      4'b1111:
79.      begin
80.          if(~Flags[2])
81.              pc<= Rx;
82.          else
83.              pc= pc+1'b1;
84.          end
85.      4'b0000:
86.          begin pc= pc; end //Codificación
87.          default: pc<=pc+1'b1;
88.          endcase
89.      end
90.      assign Address_Instruction_Bus=pc;
91.
92. endmodule

```

Banco_R

```

1. timescale 1ns / 1ps
2. /////////////////////////////////
3. // Company: Universidad Autonoma de Zacatecas
4. // Engineer: Roboticos
5. //
6. // Create Date: 18.10.2020 15:48:20
7. // Design Name: Sel_Datos
8. // Module Name: Sel_Datos
9. // Project Name: Microprocesador_RISC_8bits
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. ///////////////////////////////

```

```

21.
22.
23. module Banco_R(
24.
25.     input [5:0] Sel_R, // Tres bits para cada uno,Sel_R[2:0] para Rx y Sel_R[5:3] para
   Ry
26.     input [7:0] Dato_N,// Dato de entrada que sera almacenado
27.     output[7:0] Rx,
28.     output[7:0] Ry,
29.     input Sel_RLE, // Con 0 va a leer y con 1 a escribir
30.     input Rst,
31.     input Clk
32. );
33.
34. reg [7:0] Registro [7:0]; //arreglo, registro de 8 bits y nos dice que son 8 registros
35.
36. always @(posedge Clk, posedge Rst) begin
37.     if(Rst)
38.         begin
39.             Registro[0]<=0;
40.             Registro[1]<=0;
41.             Registro[2]<=0;
42.             Registro[3]<=0;
43.             Registro[4]<=0;
44.             Registro[5]<=0;
45.             Registro[6]<=0;
46.             Registro[7]<=0;
47.         end
48.     else
49.         if (Sel_RLE)
50.             Registro[Sel_R[2:0]]<=Dato_N;//se esta escribe lo que tenga Dato_N en
   Rx
51.
52.     end
53.
54.     assign Ry=Registro[Sel_R[5:3]];
55.     assign Rx=Registro[Sel_R[2:0]];
56. endmodule

```

Control_Sali

```

1. timescale 1ns / 1ps
2.///////////
3.// Company:
4.// Engineer:
5.//
6.// Create Date: 10.11.2020 22:52:42

```

```

7.// Design Name:
8.// Module Name: Control_Sali
9.// Project Name:
10.// Target Devices:
11.// Tool Versions:
12.// Description:
13.//
14.// Dependencies:
16.//
17.// Revision:
18.// Revision 0.01 - File Created
19.// Additional Comments:
20.//
21.///////////
22.
23.module Control_Sali(
24.    input [7:0] Ry,
25.    input [7:0] Rx,
26.    input [7:0] Num,
27.    input Clk,
28.    input Rst,
29.    input [1:0] Sel_Sali,
30.    output reg [7:0] Address_Data_Bus,
31.    output reg [7:0] DataOut_Bus,
32.    output reg L_E
33.);

34.
35.always @(posedge Clk, posedge Rst)
36.begin
37.if (Rst)
38.begin
39.Address_Data_Bus<=0;
40.DataOut_Bus<=0;
41.L_E<=0;
42.end
43.else
44.case (Sel_Sali)
45.2'b00: begin //Nop
46.Address_Data_Bus<=0;
47.DataOut_Bus<=0;
48.L_E<=0; end
49.2'b01: begin// Load [Ry], Rx
50.Address_Data_Bus<=Ry;
51.    DataOut_Bus<=0;
52.L_E<=0; //Lectura
53.end
54.2'b10: begin //Store [Rx],#Num
55.Address_Data_Bus<=Rx;

```

```

56. DataOut_Bus<=Num;
57. L_E<=1; //Escritura
58.end
59.2'b11: begin //Store [Rx],Ry
60. Address_Data_Bus<=Rx;
61.DataOut_Bus<=Ry;
62.L_E<=1; //Escritura
63.end
64.default: begin
65.Address_Data_Bus<=0;
66.DataOut_Bus<=0;
67.L_E<=0;//Lectura
68.end
69.endcase
70.end
71.always @((Rx,Ry,Num,Sel_Sali) begin
72.if (Sel_Sali==2'b01)
73.begin
74. DataOut_Bus<=0;
75.Address_Data_Bus<=Ry;
76.end
77.end
78.endmodule

```

Memoria_RAM

```

1.`timescale 1ns / 1ps
2///////////
3// Company:
4// Engineer:
5//
6// Create Date: 11.11.2020 09:11:48
7// Design Name:
8// Module Name: Memoria_Ram
9// Project Name:
10// Target Devices:
11// Tool Versions:
12// Description:
13//
14// Dependencies:
15//
16// Revisión:
17// Revision 0.01 - File Created
18// Additional Comments:
19//
20///////////
21.
22.

```

```

23.module Memoria_RAM
24.#(parameter Ld=256, m=8)(
25.input L_E,
26.input [m-1:0] address_data,
27.input [m-1:0] data_in,
28.output reg [m-1:0] N_dataout
29.);

30.reg [(m-1):0]mem[0:(Ld-1)];
31.
32.initial
33.begin
34.$readmemh("RAM.mem",mem);
35.end
36.
37.always @(address_data,data_in,L_E) begin
38.if(L_E)
39.    begin
40.        mem[address_data]<=data_in;
41.        N_dataout<=mem[address_data];
42.end
43.else
44.N_dataout<=mem[address_data];
45.end
46.endmodule

```

Memoria_ROM

```

1.`timescale 1ns / 1ps
2.///////////////////////////////
3.// Company:
4.// Engineer:
5.//
6.// Create Date: 11.11.2020 09:11:48
7.// Design Name:
8.// Module Name: Memoria_ROM
9.// Project Name:
.10.// Target Devices:
11.// Tool Versions:
12.// Description:
13.//
14.// Dependencies:
15.//
16.// Revision:
17..// Revision 0.01 - File Created
18..// Additional Comments:

```

```

19.//  

20.//////////  

21.  

22.  

23.module Memoria_ROM  

24.#(parameter Ld=256, m=8, n=9)(  

    25.input [m-1:0] address,  

    26.output reg [n-1:0] N_dout  

    27);  

28.reg [(n-1):0]mem[0:(Ld-1)];  

29.  

30.initial  

31.begin  

32 .$readmemb("ROM.mem",mem);  

33.end  

34.  

35.always @(address) begin  

36.N_dout<=mem[address];  

37.end  

38. endmodule

```

Multiplicación

001001100
 001010010
 001011001
 001100100
 101000101
 110001000
 101101000
 101000010
 110011001
 101010000
 111100011
 101000101

División

001001100
 001010010
 001011001
 001100100
 101000101
 110011000
 101101000
 101000001
 110010001

```
101001000
111100111
101000101
110011001
```

Apéndice B. Código de tests benches

Tb_MicroMem

```

1. `timescale 1ns / 1ps
2. ///////////////////////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 11.11.2020 10:06:47
7. // Design Name:
8. // Module Name: tb_MicroMem
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. ///////////////////////////////////////////////////////////////////
21.
22.
23.module tb_MicroMem;
24.
25.reg Clk;
26. reg Rst;
27.
28. MicroMem uut(
29.   .Clk(Clk),
30.   .Rst(Rst)
31. );
```

```

32.
33. initial
34. begin
35. Rst=1;
36. Clk=0;
37.
38. #2 Rst=0;
39. #10 $finish;
40. end
41.
42. always
43. #1 Clk = !Clk;
44.endmodule

```

Tb_Microprocesador_8bits

```

1.`timescale 1ns / 1ps
2.///////////////////////////////
3.// Company:
4.// Engineer:
5.//
6.// Create Date: 10.11.2020 23:13:24
7.// Design Name:
8.// Module Name: Tb_Microprocesador_RISC_8bits
9.// Project Name:
10.// Target Devices:
11// Tool Versions:
12// Description:
13.//
14.// Dependencies:
15.//
16.// Revision:
17.// Revision 0.01 - File Created
18.// Additional Comments:
19.//

```

```

20.///////////
21.
22.
23.module Tb_Microprocesador_RISC_8bits;
24.reg Clk;
25.reg Rst;
26.reg [7:0] DataIn_Bus;
27.reg [8:0] Instruction;
28.wire [7:0] Address_Data_Bus;
29.wire [7:0] DataOut_Bus;
30.wire L_E;
31.wire [7:0] Address_Instruction_Bus;

35.Microprocesador_RISC_8bits uut(
36..Clk(Clk),
37..Rst(Rst),
38..Instruction(Instruction),
39..DataIn_Bus(DataIn_Bus),
40..Address_Instruction_Bus(Address_Instruction_Bus),
41..DataOut_Bus(DataOut_Bus),
42..Address_Data_Bus(Address_Data_Bus),
43..L_E(L_E)
44.);

45.initial
46.begin
47.Rst=1;
48.Clk=0;
49.Instruction=0;
50.DataIn_Bus=0;
51.
52.#1 Rst=0; Instruction=9'b001_000_111; //load Rx Num
53.#2 Instruction=9'b001_001_110; //load Rx Num
54.#2 Instruction=9'b010_010_000; // load Rx [Ry]
55.DataIn_Bus=10;
56.#2 Instruction=9'b011_001_010; // store [Rx] Num
57.#2 Instruction=9'b100_010_000; // store [Rx] Ry
58.#2 Instruction=9'b101_011_000; // move Rx Ry
59.#2 Instruction=9'b110_011_001; // Math Rx OP
60.#2 Instruction=9'b111_100_001; // Jump [Rx] Cond
61.#2 Instruction=9'b000_100_001; // Nop
62.
63.end
64.
65.always
66.#1 Clk = !Clk;

```

67.
68.
69.endmodule

Tb_ALU

```
1.`timescale 1ns / 1ps
2.///////////
3.// Company:
4.// Engineer:
5.//
6.// Create Date: 10.11.2020 19:34:30
7.// Design Name:
8.// Module Name: Tb_ALU
9.// Project Name:
10.// Target Devices:
11.// Tool Versions:
12.// Description:
13.//
14.// Dependencies:
15.//
16.// Revision:
17.// Revision 0.01 - File Created
18.// Additional Comments:
19.//
20.///////////
21.
22.
23.module Tb_ALU;
24. reg[2:0]Ope;
25. reg [7:0]Ry;
26. reg [7:0]Rx;
27. wire [7:0] Resultado;
```

```

28. wire[2:0] Flags;
29.
30. ALU uut (
31..Ope(Ope),
32 .Ry(Ry),
33 .Rx(Rx),
34.Resultado(Resultado),
35. .Flags(Flags));

36. initial
37. begin
38. Ry=8'b00000011;
39. Rx=8'b00000100;
40. Ope=0;
41. #2 Ope=3'b000;
42. #2 Ope=3'b001;
43. #2 Ope=3'b100;
44.
45. end
46.endmodule

```

Tb_Conca_Num

```

1.`timescale 1ns / 1ps
2.///////////
3.// Company:
4.// Engineer:
5.//
6.// Create Date: 10.11.2020 23:08:14
7.// Design Name:
8.// Module Name: Tb_Conca_Num
9.// Project Name:
10.// Target Devices:
11.// Tool Versions:
12.// Description:

```

```

13.//  

14.// Dependencies:  

15.//  

16.// Revision:  

17.// Revision 0.01 - File Created  

18.// Additional Comments:  

19.//  

20.///////////  

21.  

22.  

23.module Tb_Conca_Num;  

24.  

25.    reg [7:0] Instruction;  

26.    wire [7:0] Num;  

27.    Conca_Num uut(  

28.        .Instruction(Instruction),  

29.        .Num(Num)  

30.    );  

31.    initial  

32.    begin  

33.        Instruction = 0;  

34.        35.  

36.        37.#2Instruction = 8'b00000001;  

38.  

39.  

40.#2Instruction = 8'b00001011;  

41.  

42.  

43.#2Instruction = 8'b11111111;  

44. end  

45.endmodule

```

Tb_Banco_R

```

1. `timescale 1ns / 1ps  

2. //////////////////////////////////////////////////////////////////  

3. // Company: Universidad Autonoma de Zacatecas  

4. // Engineer: Roboticos  

5. //  

6. // Create Date: 18.10.2020 15:48:20  

7. // Design Name: Sel_Datos  

8. // Module Name: Sel_Datos  

9. // Project Name: Microprocesador_RISC_8bits

```

```
10. // Target Devices:  
11. // Tool Versions:  
12. // Description:  
13. //  
14. // Dependencies:  
15. //  
16. // Revision:  
17. // Revision 0.01 - File Created  
18. // Additional Comments:  
19. //  
20. ////////////////////////////////  
  
21.  
22.  
23.module Tb_Banco_R;  
24. reg Rst;  
25. reg Clk;  
26. reg [5:0] Sel_R;  
27. reg Sel_RLE;  
28. reg [7:0] Dato_N;  
29. wire [7:0] Rx;  
30. wire [7:0] Ry;  
31.  
32. Banco_R uut(  
33. .Rst(Rst),  
34. .Clk(Clk),  
35. .Sel_R(Sel_R),  
36. .Sel_RLE(Sel_RLE),  
37. .Dato_N(Dato_N),  
38. .Rx(Rx),  
39. .Ry(Ry));  
40.  
41. initial  
42. begin  
43. Rst=1;
```

```

44. Clk=0;
45. Sel_R=0;
46. Sel_RLE=0;
47. Dato_N=0;
48. #2 Rst=0; Dato_N=8'b00000000; Sel_R=6'b000_000; Sel_RLE=0;
49. #2 Dato_N=8'b00000001; Sel_R=6'b000000; Sel_RLE=1;
50. #2 Dato_N=8'b00000010; Sel_R=6'b000001; Sel_RLE=1;
51. #2 Dato_N=8'b00000011; Sel_R=6'b000010; Sel_RLE=1;
52. #2 Dato_N=8'b00000100; Sel_R=6'b000011; Sel_RLE=1;
53. #2 Dato_N=8'b00000101; Sel_R=6'b000100; Sel_RLE=1;
54. #2 Dato_N=8'b00000110; Sel_R=6'b000101; Sel_RLE=1;
55. #2 Dato_N=8'b00000111; Sel_R=6'b000110; Sel_RLE=1;
56. #2 Dato_N=8'b00001000; Sel_R=6'b000111; Sel_RLE=1;
57.
58. #2 Sel_R=6'b001000; Sel_RLE=0;
59. #2 Sel_R=6'b011010; Sel_RLE=0;
60. #2 Sel_R=6'b101100; Sel_RLE=0;
61. #2 Sel_R=6'b111110; Sel_RLE=0;
62.
63. end
64.
65. always
66.#1 Clk = !Clk;
67.
68.endmodule

```

Tb_Control_Sali

```

1. `timescale 1ns / 1ps
2. ///////////////////////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 10.11.2020 22:54:27

```

```
7. // Design Name:  
8. // Module Name: Tb_Control_Sali  
9. // Project Name:  
10. // Target Devices:  
11. // Tool Versions:  
12. // Description:  
13. //  
14. // Dependencies:  
15. //  
16. // Revision:  
17. // Revision 0.01 - File Created  
18. // Additional Comments:  
19. //  
20. /////////////////////////////////  
  
21.  
22.  
23.module Tb_Control_Sali;  
24. reg [7:0] Ry;  
25. reg [7:0] Rx;  
26. reg [7:0] Num;  
27. reg [1:0] Sel_Sali;  
28. wire [7:0] Address_Data_Bus;  
29. wire [7:0] DataOut_Bus;  
30. wire L_E;  
31. reg Clk;  
32. reg Rst;  
33.  
34. Control_Sali uut(  
35. .Ry(Ry),  
36. .Rx(Rx),  
37. .Num(Num),  
38. .Sel_Sali(Sel_Sali),  
39. .Address_Data_Bus(Address_Data_Bus),  
40. .DataOut_Bus(DataOut_Bus),  
41. .L_E(L_E),  
42. .Clk(Clk),
```

```
43. .Rst(Rst)
44.      );
45.
46. initial
47. begin
48.   Rst=1;
49.   Ry=8'b11011010;
50.   Rx=8'b00110010;
51.   Num=8'b10010111;
52.   Sel_Sali=2'b01;
53.
54. #2
55.   Clk=1;
56.   Sel_Sali=2'b01;
57.
58. #2
59.   Sel_Sali=2'b11;
60.   Clk=1;
61. #2
62.   Sel_Sali=2'b10;
63.   Clk=1;
64. #2
65.   Sel_Sali=2'b00;
66.   Clk=1;
67. end
68.endmodule
```