Universidad Autónoma de Zacatecas

Unidad Académica de Ingeniería Eléctrica

Programa Académico

de Ingeniería en Robótica y Mecatrónica

Microcontroladores

ARQUITECTURA MICRO_UAZ_8BITS_2020

Docente:

Dr. Remberto Sandoval Aréchiga

Equipo: Patriarcado Team

Hernández Pérez José Humberto

Salazar Ortega Enrique

Castro Pérez Juan Uriel

Martínez Quiroz Jesús

Ruiz Carrillo Ulises

Grupo

5to "A"

Fecha de Entrega: 18 noviembre 2020

Índice

Resumen

El microcontrolador elaborado en este proyecto fue elaborado a partir de un diagrama de caja negra y del set de instrucciones ya definidos. A partir del análisis de cada una de las instrucciones que componen al set de instrucciones es que comenzamos a definir cada uno de los componentes necesarios para poder realizar dichas instrucciones, una vez deducidos los componentes necesarios establecimos las conexiones entre los buses que unen cada componente y de esta manera obtener un diagrama de caja blanca que sea capaz de cumplir con lo establecido en el set de instrucciones.

El siguiente paso que se realizó, fue definir los diagramas de caja negra y descripción funcional, así como las descripciones de entradas y salidas de cada uno de los componentes que conforman a nuestro microcontrolador siendo estos la unidad de control, el banco de registros, la unidad aritmética lógica, unidad de saltos, control de bus, un multiplexor y un bloque llamado ConNum. Posteriormente se realizó la codificación en el software de vivado a partir de la descripción funcional, así mismo elaboramos los bancos de pruebas para verificar que cada uno de los componentes cumpliera con su función. Finalmente se realizó el código top del diseño donde se conectan todos los buses de los componentes y se provo su funcionamiento mediante un banco de pruebas de todo el microcontrolador

Introducción

Microcontroladores

Es un circuito integrado de alta escala de integración que incorpora la mayor parte de los elementos que configuran un controlador y que contiene todos los componentes fundamentales de un ordenador, aunque de limitadas prestaciones y que se suele destinar a gobernar una sola tarea. En su memoria sólo reside un programa que controla en funcionamiento de una tarea determinada, sus líneas de entrada/salida se conectan a los sensores y actuadores del dispositivo a controlar y, debido a su pequeño tamaño, suele ir integrado en el propio dispositivo al que gobierna.

Arquitectura Harvard

La organización del computador según el modelo Harvard, básicamente, se distingue del modelo Von Neumann por la división de la memoria en una memoria de instrucciones y una memoria de datos, de manera que el procesador puede acceder separada y simultáneamente a las dos memorias.

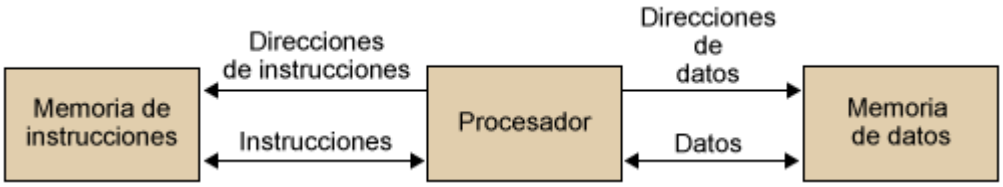


Figura 1. Arquitectura Harvard.

El procesador dispone de un sistema de conexión independiente para acceder a la memoria de instrucciones y a la memoria de datos. Cada memoria y cada conexión pueden tener características diferentes; por ejemplo, el tamaño de las palabras de memoria (el número de bits de una palabra), el tamaño de cada memoria y la tecnología utilizada para implementarlas.

Instrucción RISC

Se trata de un tipo de procesador especialmente rápido que utiliza una tecnología del tipo pipeline muy desarrollada, lo que le faculta para operar con un alto nivel de simultaneidad. Este tipo de procesadores son lo contrario de los denominados CISC, mucho más comunes. Las características comunes a todos los procesadores RISC, fuente de sus capacidades de altas prestaciones son: Modelo de conjunto de instrucciones Load/Store que significa: Cargar-Almacenar. Sólo las instrucciones Load/Store acceden a memoria; las demás operaciones en un RISC, tienen lugar en su gran conjunto de registros. Ello simplifica el direccionamiento y acorta los tiempos de los ciclos de la CPU.

Sistema digital

Un sistema digital es cualquier sistema que permita crear, decodificar, transmitir o guardar información que se encuentra representada en cantidades tan restringidas que sus señales de entrada y salida solo admiten valores discretos. Los valores discretos son variables que no aceptan cualquier valor, sino solo aquellos que pertenezcan a su conjunto, por tanto, son finitos. En este sentido, un sistema digital es todo dispositivo que manipule datos mediante dígitos que casi siempre están representados con el código binario. El sistema binario solo admite ceros (0) y unos (1) como valores, por lo tanto, se trata de valores discretos. Actualmente, los sistemas digitales se encuentran incorporados en dispositivos magnéticos, electrónicos y mecánicos.

FPGA

FPGA son las siglas en inglés de matriz de puertas programables en campo. Esencialmente, un FPGA es un circuito de hardware que un usuario puede programar para realizar una o más operaciones lógicas. Yendo un paso más allá, los FPGA son circuitos integrados, o IC, que son conjuntos de circuitos en un chip, esa es la parte de la "matriz". Esos circuitos, o matrices, son grupos de puertas lógicas programables, memoria u otros elementos. Los FPGA brindan beneficios a los diseñadores de muchos tipos de equipos electrónicos, que van desde redes de energía inteligente, navegación de aeronaves, asistencia al conductor de automóviles, ultrasonidos médicos y motores de búsqueda de centros de datos, solo por nombrar algunos.

Verilog

Verilog es un lenguaje de descripción de hardware (HDL, del Inglés Hardware Description Language) usado para modelar sistemas electrónicos. El lenguaje, algunas veces llamado Verilog HDL, soporta el diseño, prueba e implementación de circuitos analógicos, digitales y de señal mixta a diferentes niveles de abstracción.

Circuitos combinacionales

Estos circuitos se desarrollaron usando puertas lógicas AND, OR, NOT, NAND y NOR. Estas puertas lógicas son bloques de construcción de circuitos combinacionales. Un circuito combinacional consta de variables de entrada y variables de salida. Dado que estos circuitos no dependen de la entrada anterior para generar ninguna salida, se les llama circuitos lógicos combinacionales. Un circuito combinacional puede tener un número n de entradas y un número m de salidas. En los circuitos combinacionales, la salida en cualquier momento es una función directa de las entradas externas aplicadas.

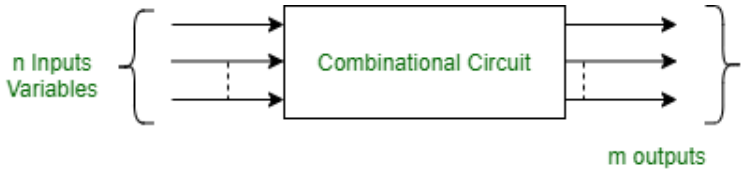


Figura 2. Circuito Combinacional.

Circuitos Secuenciales

Un circuito secuencial se especifica mediante una secuencia de tiempo de entradas, salidas y estados internos. La salida de un circuito secuencial depende no solo de la combinación de las entradas presentes sino también de las salidas anteriores. A diferencia de los circuitos combinacionales, los circuitos secuenciales incluyen elementos de memoria con circuitos combinacionales. A partir de los temas anteriormente mencionados es necesario resaltar la importancia que tiene para un ingeniero en el campo de la electrónica conocer acerca del diseño de sistemas digitales. Con la implementación de la siguiente practica se tiene como objetivo el diseñar un controlador de display de 7 segmentos aplicando la metodología top down en el diseño de diagramas, elaboración de código e implementación, para la posterior observación de su funcionamiento a través de la FPGA Basys 3.

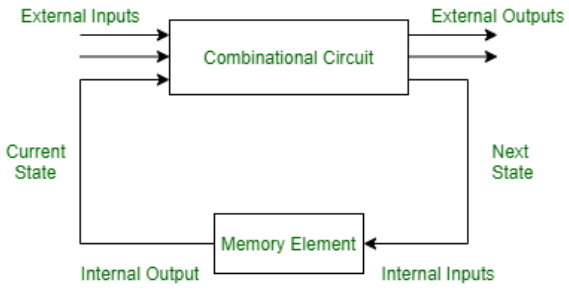


Figura 3. Circuito Secuencial.

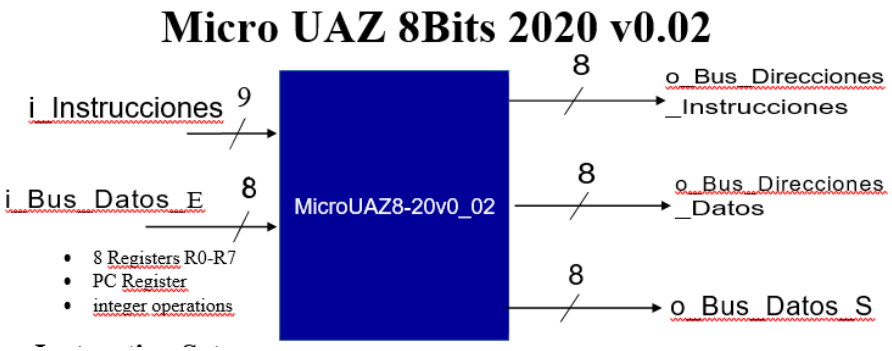


Diagrama 1. Caja Negra de Microcontrolador.

Set de Instrucciones

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>>RX 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.

Arquitectura

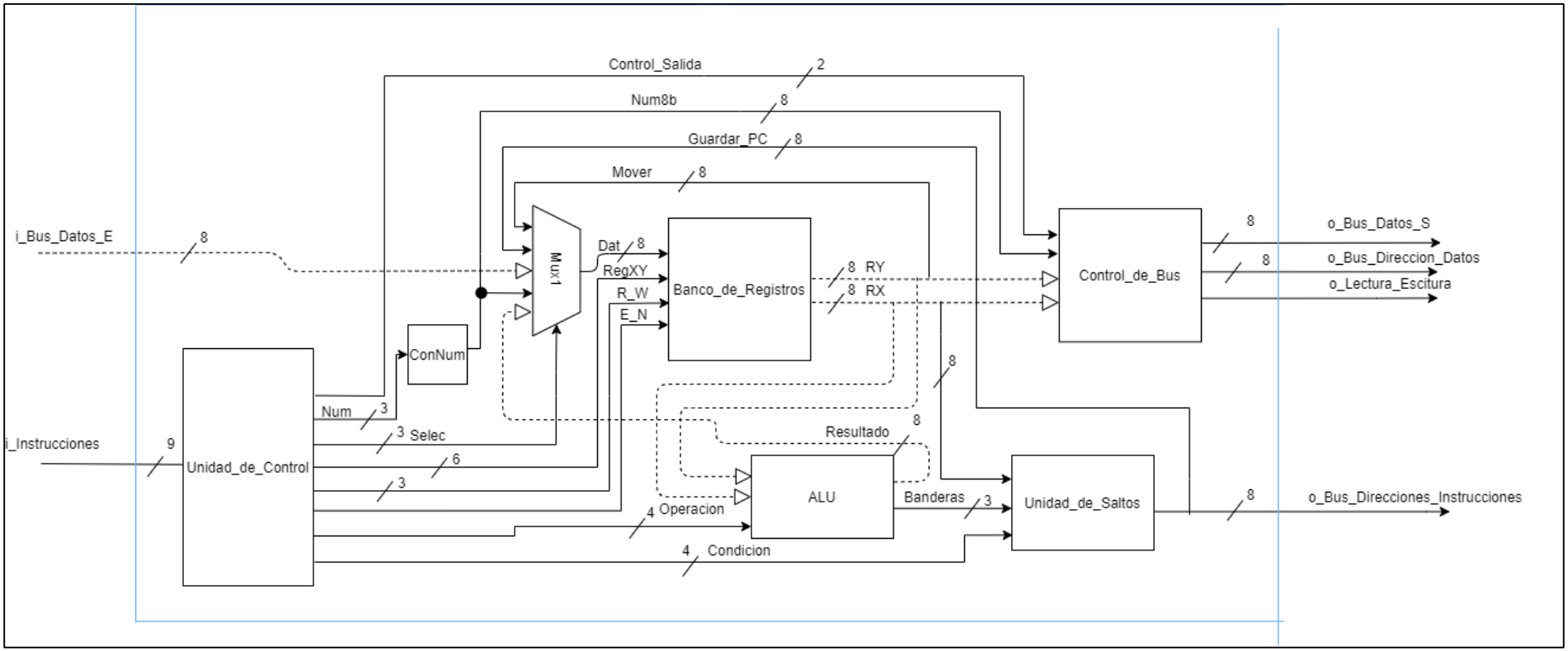


Diagrama 2. Caja Blanca de Microcontrolador.

Fecha de Entrega: 18 noviembre 2020

Unida de Control

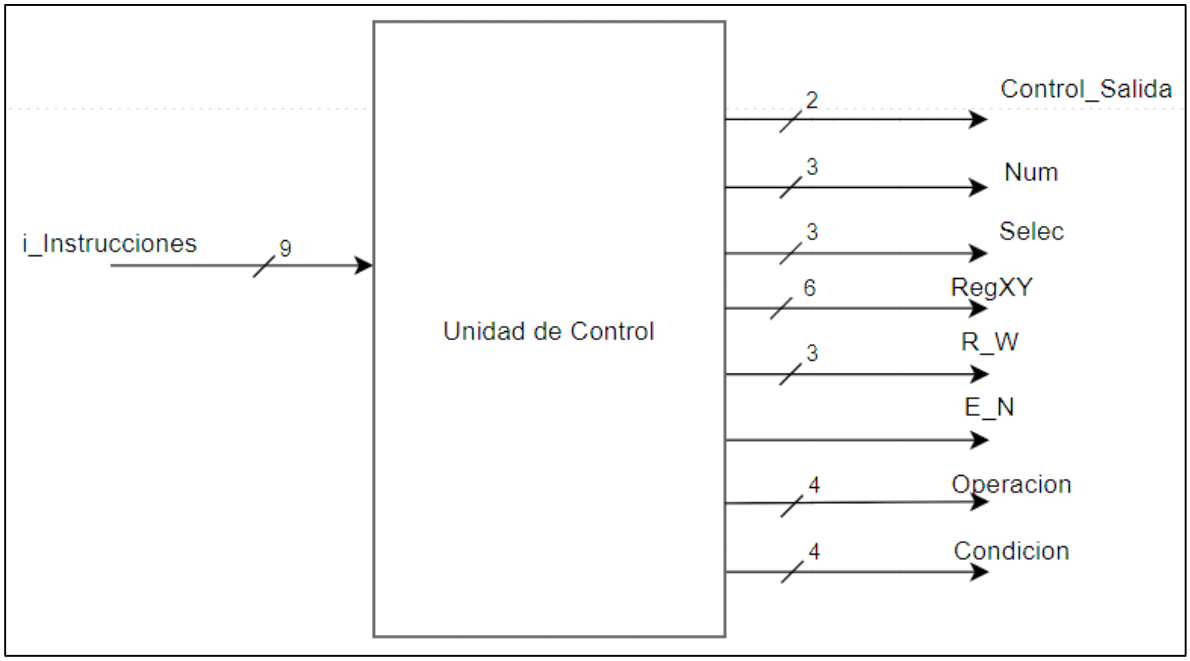


Diagrama 3. Unidad de Control.

Descripción de Señales de Entrada

Nombre de la señal	***Tamaño (bits)***	***Descripción***
i_Instrucciones	*9*	*Bus que se encarga de transferir las instrucciones desde la memoria de instrucciones para su ejecución.*

Tabla 2. Descripción de Señales de Entrada.

Fecha de Entrega: 18 noviembre 2020

Descripción de Señales de Salida

Nombre de la señal	*Tamaño (bits)*	*Descripción*
Control_Salida	*2*	*Señal de salida encargada de controlar al control de bus. Es la encargada de indicar al control de bus si los datos provenientes del banco de registros serán enviados a través de o_Bus_Datos_S o o_Bus_Direccion_Datos.*
Num	*3*	*Señal que contiene al valor de num en 3 bit.*
Selec	*3*	*Señal que permite seleccionar el dato que entra al banco de registros para escribirse en algún registro.*
RegXY	*6*	*Permite seleccionar los registros que van a ser leídos. los 3 bits más significativos determinan a RX y los 3 bit menos significativos corresponden a RY.*
R_W	*3*	*Permite seleccionar al registro en el que se va a escribir un dato.*
E_N	*1*	*Señal que controla la acción a realizar en el banco de registros ya sea hacer una escritura en un registro, mover el dato de un registro a otro o no hacer nada.*
Operación	*4*	*Señal que indica a la ALU la operación que se va a realizar mediante los 3 bit menos significativos. El bit más significativo permite habilitar al ALU*
Condicion	*4*	*Señal que indica a la Unidad de Saltos la Condición a evaluar mediante los 3 bit menos significativos. El bit más significativo permite resetear al PC.*

Tabla 3. Descripción de Señales de Salida.

Descripción Funcional

```
1     Inicio
2     //Entradas y salidas
3     Entrada [8:0] i_Instrucciones,
4     Salida [1:0] Control_Salida,
5     Salida [2:0] Num,
6     Salida [2:0] Selec,
7     Salida [5:0] RegXY,
8     Salida [2:0] R_W,
9     Salida E_N,
10    Salida [3:0] Operacion,
11    Salida [3:0] Condicion;
12
13    case(i_Instricciones[8:6] )
14    Caso 1:
15          E_N <= 1;
16          R_W <= i_Instricciones[5:3];
17          Selec <= 3'b010;
18          Num <= i_Instricciones[2:0];
19          Condicion <= 4'b0000;
20          Operacion <= 4'b0000;
21          Control_Salida <= 2'b00;
22          RegXY <= 6'b000000;
23
24    Caso 2:
25          E_N <= 1;
26          R_W <= i_Instricciones[5:3];
27          Selec <= 3'b001;
28          Num <= 3'b000;
29          Condicion <= 4'b0000;
30          Operacion <= 4'b0000;
31          Control_Salida <= 2'b01;
32          RegXY[5:3] <= 3'b000;
33          RegXY[2:0] <= i_Instricciones[2:0];
34
35    Caso 3:
36          E_N <= 0;
37          R_W <= 3'b000;
38          Selec <= 3'b111;
```

```
39              Num <= i_Instricciones[2:0];
40              Condicion <= 4'b0000;
41              Operacion <= 4'b0000;
42              Control_Salida <= 2'b10;
43              RegXY[5:3] <= i_Instricciones[5:3];
44              RegXY[2:0] <= 3'b000;
45
46      Caso 4:
47              E_N <= 0;
48              R_W <= 3'b000;
49              Selec <= 3'b111;
50              Num <= 3'b000;
51              Condicion <= 4'b0000;
52              Operacion <= 4'b0000;
53              Control_Salida <= 2'b11;
54              RegXY <= i_Instricciones[5:0];
55
56      Caso 5:
57              E_N <= 1;
58              R_W <= i_Instricciones[5:3];
59              Selec <= 3'b100;
60              Num <= 3'b000;
61              Condicion <= 4'b0000;
62              Operacion <= 4'b0000;
63              Control_Salida <= 2'b00;
64              RegXY[5:3] <= 3'b000;
65              RegXY[2:0] <= i_Instricciones[2:0];
66
67      Caso 6:
68              E_N <= 1;
69              R_W <= 3'b000;
70              Selec <= 3'b011;
71              Num <= 3'b000;
72              Condicion <= 4'b0000;
73              Operacion[2:0] <= i_Instricciones[2:0];
74              Operacion[3] <= 1;
75              Control_Salida <= 2'b00;
76              RegXY[5:3] <= i_Instricciones[5:3];
77              RegXY[2:0] <= 3'b000;
78
79      Caso 7:
```

```
80          R_W <= 3'b111;
81          Num <= 3'b000;
82          Condicion[2:0] <= i_Instricciones[2:0];
83          Condicion[3] <= 1;
84          Operacion <= 4'b0000;
85          Control_Salida <= 2'b00;
86          RegXY[5:3] <= i_Instricciones[5:3];
87          RegXY[2:0] <= 3'b000;
88          if (i_Instricciones[2:0] == 2'b001)
89                  E_N <= 1;
90                  Selec <= 0;
91          else
92          E_N <= 0;
93          Selec <= 7;
94
95          end
96    Otro Caso
97          E_N <= 0;
98          R_W <= 3'b000;
99          Selec <= 3'b111;
100         Num <= 3'b000;
101         Condicion <= 4'b0000;
102         Operacion <= 4'b0000;
103         Control_Salida <= 2'b00;
104         RegXY <= 6'b000000;
105
106   Fin
```

Pseudocódigo 1. Unidad de Control.

Banco de Registros

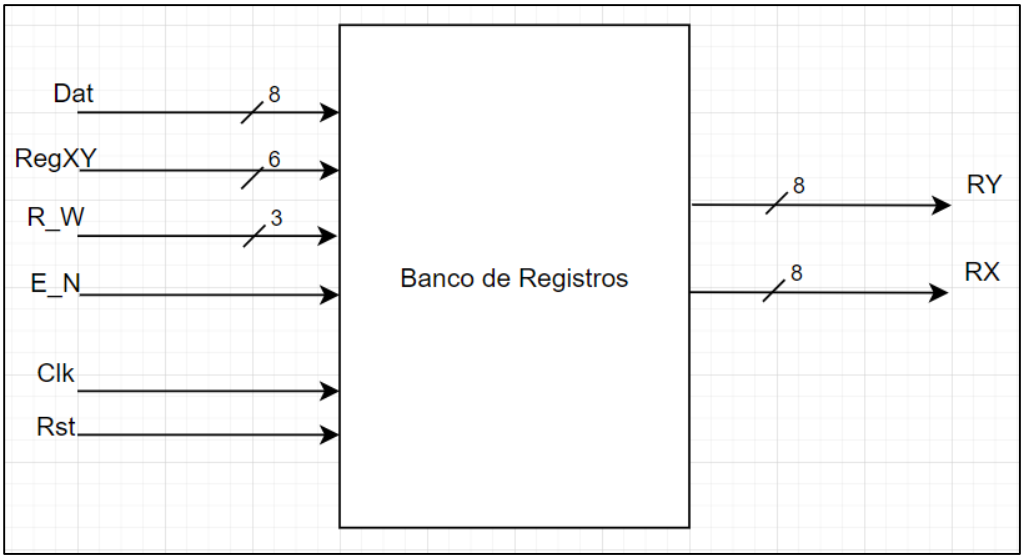


Diagrama 4. Banco de Registros.

Descripción de Señales de Entrada

Nombre de la señal	*Tamaño (bits)*	*Descripción*
Dat	*8*	*Transfiere los 8 bits que va a ser almacenados en el banco de registros.*
RegXY	*6*	*Permite seleccionar los registros que van a ser leídos. los 3 bits más significativos determinan a RX y los 3 bit menos significativos corresponden a RY.*
R_W	*3*	*Permite seleccionar al registro en el que se va a escribir un dato.*
E_N	*1*	*Señal que controla la acción a realizar en el banco de registros ya sea hacer una escritura en un registro, mover el dato de un registro a otro o no hacer nada.*
Clk	*1*	*Entrada de referencia en el tiempo, debe ser una señal periódica con una frecuencia de 100MHz.*
Rst	*1*	*Señal de Entrada que restaura el sistema a una configuración inicial de los registros, memoria, etc.*

Tabla 4. Descripción de Señales de Entrada.

Descripción de Señales de Salida

Nombre de la señal	*Tamaño (bits)*	*Descripción*
RY	*8*	*Señal de salida que permite la transferencia de datos de RY hasta la ALU.*
RX	*8*	*Señal de salida que permite la transferencia de datos de RY hasta la ALU.*

Tabla 5. Descripción de Señales de Salida

```
1    Inicio
2    //Entradas y salidas
3    Entrada Clk,
4    Entrada Rst,
5    Entrada [7:0] Dat,
6    Entrada [5:0] RegXY,
7    Entrada [2:0] R_W,
8    Entrada E_N,
9    Salida [7:0] RX,
10   Salida [7:0] RY;
11   );
12
13   reg [7:0] Registro[0:7];
14
15   Cada flanco positivo de reloj
16
17   Si Rst = 1
18   Registro[0] <= 0;
19   Registro[1] <= 0;
20   Registro[2] <= 0;
21   Registro[3] <= 0;
22   Registro[4] <= 0;
23   Registro[5] <= 0;
24   Registro[6] <= 0;
25   Registro[7] <= 0;
26
27   Si (E_N = 1)
28   Registro[R_W] <= Dat;
29
30   asignacion RX = Registro[RegXY[5:3]];
     asignacion RY = Registro[RegXY[2:0]];
     Fin
```

Pseudocódigo 2. Banco de Registros.

ALU

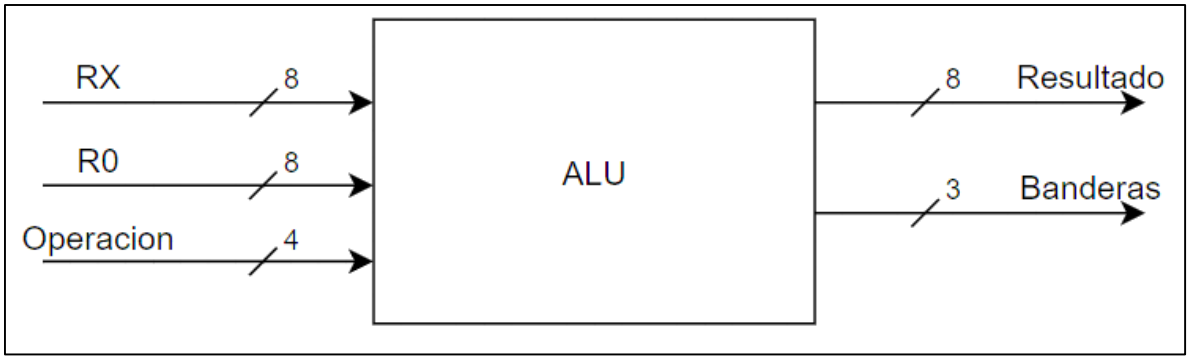


Diagrama 5. ALU

Descripción de Señales de Entrada

Nombre de la señal	Tamaño (bits)	Descripción
RX	*8*	*Señal de entrada que permite la transferencia de datos de RX hasta la ALU .*
R0	*8*	*Señal de entrada que permite la transferencia de datos de R0 hasta la ALU.*
Operacion	*4*	*Señal de entrada que indica la operación que se va a realizar con los 3 bits menos significativos y habilita la ALU a través del bit sobrante o más significativo.*

Tabla 6. Descripción de Señales de Entrada.

Descripción de Señales de Salida

Nombre de la señal	Tamaño (bits)	Descripción
Resultado	8	Señal que transporta el resultado obtenido de las operaciones realizadas por el ALU.
Banderas	3	Indica el tipo de bandera obtenida de la operación realizada en el ALU.

Tabla 7. Descripción de Señales de Salida.

Descripción Funcional

```
1   Inicio
2   //Entradas y salidas
3   Entrada [7:0] RX,
4   Entrada [7:0] R0,
5   Entrada [3:0] Operacion,
6   Salida [7:0] Resultado,
7   Salida [2:0] Banderas);
8
9   reg [8:0] AResultado =0;
10
11     Si (Operacion[3] == 1)
12        Caso (Operacion[2:0])
13           Caso 0:
14              AResultado <= R0 + RX;
15
16           Caso 1:
17              AResultado <= R0 - RX;
18
19           Caso 2:
20              AResultado[7:0] <= R0 << RX;
```

```
21              Caso 3:
22                 AResultado <= R0 >> RX;
23
24              Caso 4:
25                 AResultado <= ~ RX;
26
27              Caso 5:
28                 AResultado <= R0 & RX;
29
30              Caso 6:
31                 AResultado <= R0 | RX;
32
33              Caso 7:
34                 AResultado <= R0 ^ RX;
35
36           Fin del caso
37        De lo contrario
38           AResultado <= AResultado;
39
40        asignación Resultado = AResultado[7:0];
41        asignación Banderas[0] = &(~AResultado);
42        asignación Banderas[1] = AResultado[8];
          asignación Banderas[2] = AResultado[7];
     Fin
```

Pseudocódigo 3. ALU.

Unidad de Saltos

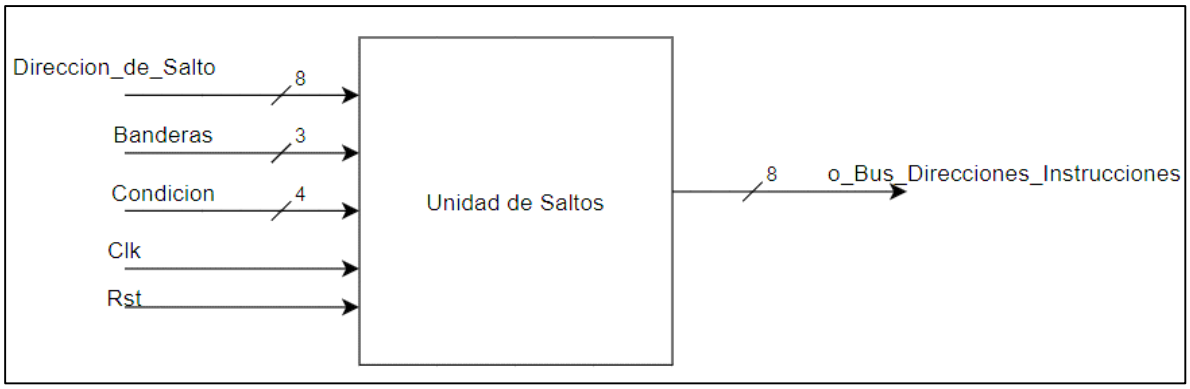


Diagrama 6. Unidad de Saltos.

Descripción de Señales de Entrada

Nombre de la señal	*Tamaño (bits)*	*Descripción*
Clk	*1*	*Entrada de referencia en el tiempo, debe ser una señal periódica con una frecuencia de 100MHz.*
Rst	*1*	*Señal de Entrada que restaura el sistema a una configuración inicial de los registros, memoria, etc.*
Direccion_de_salto	*8*	*Señal por la cual se transfieren los datos de RX hacia la unidad de saltos.*
Banderas	*3*	*Indica el tipo de bandera obtenida de la operación realizada en el ALU.*
Condicion	*4*	*Señal que indica a la Unidad de Saltos la Condición a evaluar mediante los 3 bit menos significativos. El bit más significativo permite habilitar a la Unidad de Saltos.*

Tabla 8. Descripción de Señales de Entrada.

Descripción de Señales de Salida

Nombre de la señal	*Tamaño (bits)*	*Descripción*
o_Bus_Direcciones_Instrucciones	*8*	*Señal que se encarga de enviar el contenido del PC (dirección de la siguiente instrucción a ejecutar), a la memoria de instrucciones.*

Tabla 9. Descripción de Señales de Salida

Descripción Funcional

```
1   Inicio
2   //Entradas y salidas
3   Entrada Clk,
4   Entrada Rst,
5   Entrada [7:0] Direccion_de_Salto,
6   Entrada [2:0] Banderas,
7   Entrada [3:0] Condicion,
8   Salida reg [7:0] o_Bus_Direcciones_Instrucciones;
9   );
10
11  Cada flanco positivo de reloj
12      Si Rst =1
13          o_Bus_Direcciones_Instrucciones <= 0;
14
15      Si no
16        Si (Condicion[3] == 1)
17            case (Condicion[2:0])
18                Caso 0:
19                  o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
20
21                Caso 1:
22                  o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
23
24                Caso 2:
25                      Si(Banderas[0] =1)
26                          o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
27                      Si no
28                          o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
29
30                Caso 3:
31                      Si (Banderas[0]=0)
32                          o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
33                      Si no
34                          o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
35
36                Caso 4:
37                      Si(Banderas[1]=1)
38                          o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
39                      Si no
```

```
41                      o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
42              Caso 5:
43                  Si(Banderas[1]=0)
44                      o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
45                  Si no
46                      o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
47
48              Caso 6:
49                  Si(Banderas[2]=1)
50                      o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
51                  Si no
52                      o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
53
54              Caso 7:
55                  Si(Banderas[2]=0)
56                      o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
57                  Si no
58                      o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
59
60          Fin del Caso
61      Si no
62          o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
63  Fin
```

Pseudocódigo 4. Unidad de Saltos.

Control de Bus

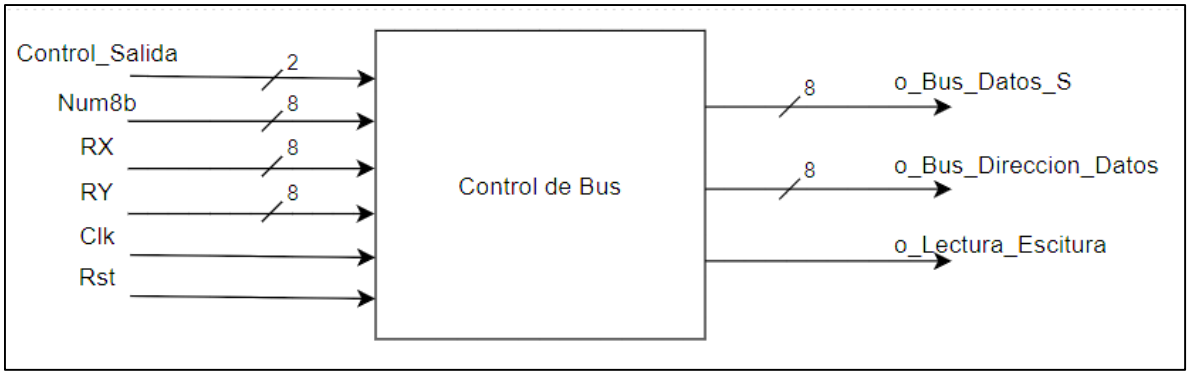


Diagrama 7. Control de Bus.

Descripción de Señales de Entrada

Nombre de la señal	Tamaño (bits)	Descripción
Clk	*1*	*Entrada de referencia en el tiempo, debe ser una señal periódica con una frecuencia de 100MHz.*
Rst	*1*	*Señal de Entrada que restaura el sistema a una configuración inicial de los registros, memoria, etc.*
Control_Salida	*2*	*Señal de salida encargada de controlar al control de bus. Es la encargada de indicar al control de bus si los datos provenientes del banco de registros serán enviados a través de o_Bus_Datos_S o o_Bus_Direccion_Datos.*
Num8b	*8*	*Transporta el valor de num contenido en 8 bits*

RX	*8*	*Señal de entrada que permite la transferencia de datos de RX hasta el control de bus*
RY	*8*	*Señal de entrada que permite la transferencia de datos de RY hasta el control de bus*

Tabla 10. Descripción de Señales de Entrada.

Descripción de Señales de Salida

Nombre de la señal	Tamaño (bits)	Descripción
o_Bus_Datos_S	*8*	*Señal de salida encargada de enviar datos hacia a la memoria de datos*
o_Bus_Direccion_Datos	*8*	*Señal de salida que se encarga de enviar la dirección de memoria*
o_Lectura_Escritura	*1*	*Controla la escritura o lectura en la memoria*

Tabla 11. Descripción de Señales de Salida.

```
1   Inicio
2   //Entradas y salidas
3   Entrada Clk,
4   Entrada Rst,
5   Entrada [1:0] Control_Salida,
6   Entrada [7:0] Num8b;
7   Entrada [7:0] RX,
8   Entrada [7:0] RY,
9   Salida  reg  [7:0] o_Bus_Datos_S,
10  Salida  reg  [7:0] o_Bus_Direccion_Datos,
11  Salida reg o_Lectura_Escitura);
12
13  Cada flanco positivo de reloj
14  Si Rst = 1
15  o_Bus_Datos_S <= 0;
16  o_Bus_Direccion_Datos <= 0;
17  o_Lectura_Escritura <= 0;
18  De lo contrario
19     Caso (Control_Salida)
20        Caso 0: //No Operacion
21           o_Bus_Datos_S <= 0;
22           o_Bus_Direccion_Datos <= 0;
23           o_Lectura_Escritura <= 0;
24
25        Caso 1: // LOAD_Registro
26           o_Bus_Datos_S <= 0;
27           o_Bus_Direccion_Datos <= RY;
28           o_Lectura_Escritura <= 0;
29        end
30        Caso 2: // STORE_Num
31           o_Bus_Datos_S <= Num8b;
32           o_Bus_Direccion_Datos <= RX;
33           o_Lectura_Escritura <= 1;
34        end
35        Caso 3: // STORE_Registro
36           o_Bus_Direccion_Datos <= RX;
37           o_Bus_Datos_S <= RY;
38           o_Lectura_Escritura <= 1;
39        end
40        Otro caso
41           o_Bus_Datos_S <= 0;
42           o_Bus_Direccion_Datos <= 0;
43           o_Lectura_Escritura <= 0;
44     Fin del caso
45   end
46  end
47  Fin
```

Pseudocódigo 5. Control de Bus.

Mux1

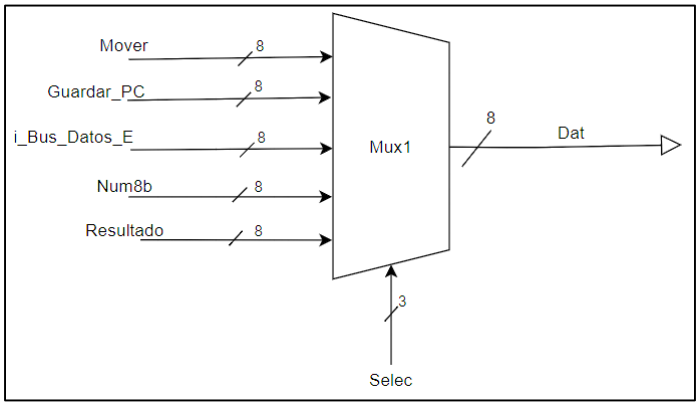


Diagrama 8. Mux1.

Descripción de Señales de Entrada

Nombre de la señal	*Tamaño (bits)*	*Descripción*
Mover	*8*	*Transfiere los datos de RY para ser almacenados en RX*
Guardar_PC	*8*	*In Transfiere los datos almacenados en el PC para ser almacenados en R7.*
i_Bus_Datos_E	*8*	*Transfiere los datos provenientes de la memoria de datos a los registros*
Num8b	*8*	*Contiene al número que va a ser almacenado de forma inmediata en los registros*
Resultado	*8*	*Señal que transporta el resultado obtenido de las operaciones realizadas por el ALU.*
Selec	*3*	*Señal que permite seleccionar el dato que entra al banco de registros para escribirse en algún registro.*

Tabla 12. Descripción de Señales de Entrada.

Descripción de Señales de Salida

Nombre de la señal	*Tamaño (bits)*	*Descripción*
Dat	*8*	*Transfiere los 8 bits que va a ser almacenados en el banco de registros.*

Tabla 13. Descripción de Señales de Salida

Descripción Funcional

```
1   Inicio
2   //Entradas y salidas
3   Entrada [7:0] Resultado,
4   Entrada [7:0] Num8b,
5   Entrada [7:0] i_Bus_Datos_E,
6   Entrada [7:0] Guardar_PC,
7   Entrada [7:0] Mover,
8   Entrada [2:0] Selec,
9   Salida [7:0] Dat;
10
11  Caso (Selec)
12  Caso 0:
13      Dat <= Guardar_PC;
14  Caso 1:
15      Dat <= i_Bus_Datos_E;
16  Caso 2:
17      Dat <= Num8b;
18  Caso 3:
19      Dat <= Resultado;
20  Caso 4:
21      Dat <= Mover;
22  Otro caso:
23      Dat <= 0;
24  Fin del cso;
25  Fin
```

Pseudocódigo 6. Mux1.

ConNum

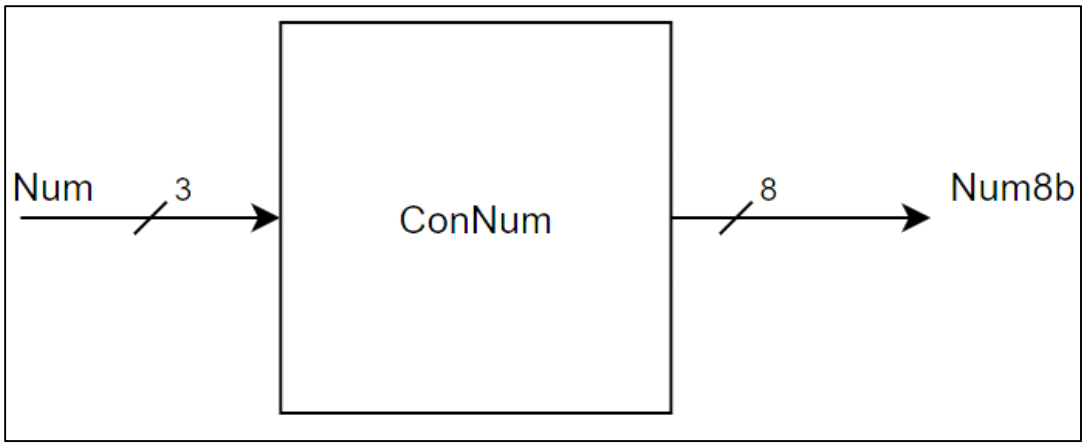


Diagrama 9. ConNum.

Descripción de Señales de Entrada

Nombre de la señal	Tamaño (bits)	Descripción
Num	*3*	*Señal que contiene al valor de num en 3 bit.*

Tabla 14. Descripción de Señales de Entrada.

Descripción de Señales de Salida

Nombre de la señal	Tamaño (bits)	Descripción
Num8b	*8*	*Transporta el valor de num contenido en 8 bits*

Tabla 15. Descripción de Señales de Salida.

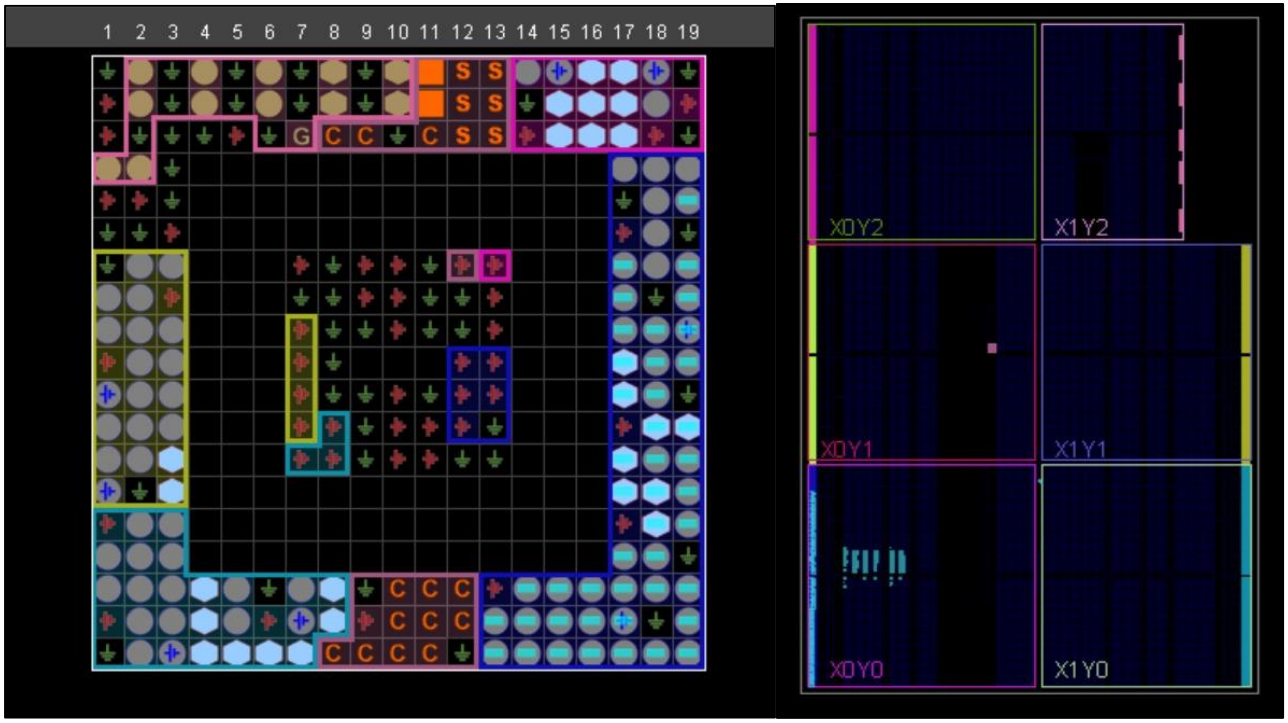
```
1   Inicio
2   //Entradas y salidas
3   Entrada [2:0] Num,
4   Salida [7:0] Num8b;
5
6   Reg [4:0] Extencion= 5'b00000;
7
8   assign Num8b <= {Extencion, Num};
9
10  Fin
```

Pseudocódigo 7. ConNum.

Implementación y Simulación

Microcontrolador de 8 Bits

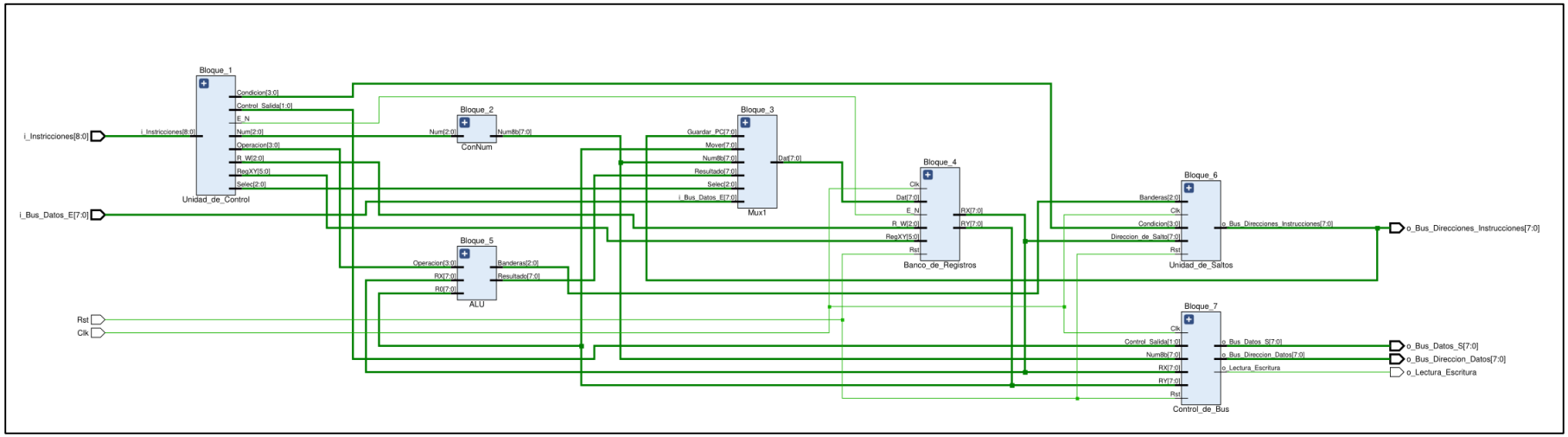
Una vez que vivado nos notifica que no existen errores en la codificación del miccontrolador, colocando nuestro cursor hacia la izquierda podemos observar el icono de run implementación, donde al seleccionarlo nos muestra la implementación de nuestro microcontrolador de 8 bits.



Implementación 1. Microcontrolador de 8 Bits.

Fecha de Entrega: 18 noviembre 2020

El esquema muestra el diagrama de caja negra que nos proporciona vivado en el apartado de esquemático, en el cual se logra apreciar cada uno de los bloques que internamente constituyen al microcontrolador de 8 bits, así mismo sus señales de entrada, salida y conexiones que lo conforman.

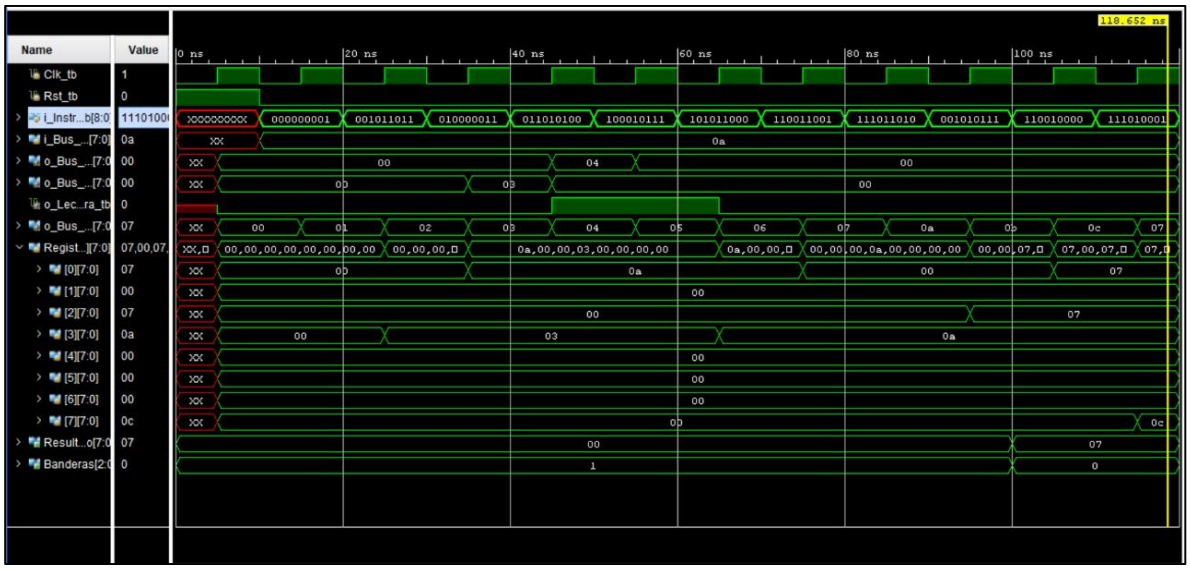


Esquema 1. Microcontrolador de 8 Bits.

Después de haber realizado la implementación del microcontrolador de 8 bits, nos dirigimos al apartado de project summary, deslizando la barra de desplazamiento verticalmente hasta su límite nos muestra en la parte inferior la gráfica y tabla de utilización.

Resource	Utilization	Available	Utilization %
LUT	223	20800	1.07
FF	89	41600	0.21
IO	44	106	41.51
BUFG	1	32	3.13

Tabla 16. Utilización del Microcontrolador de 8 Bits.

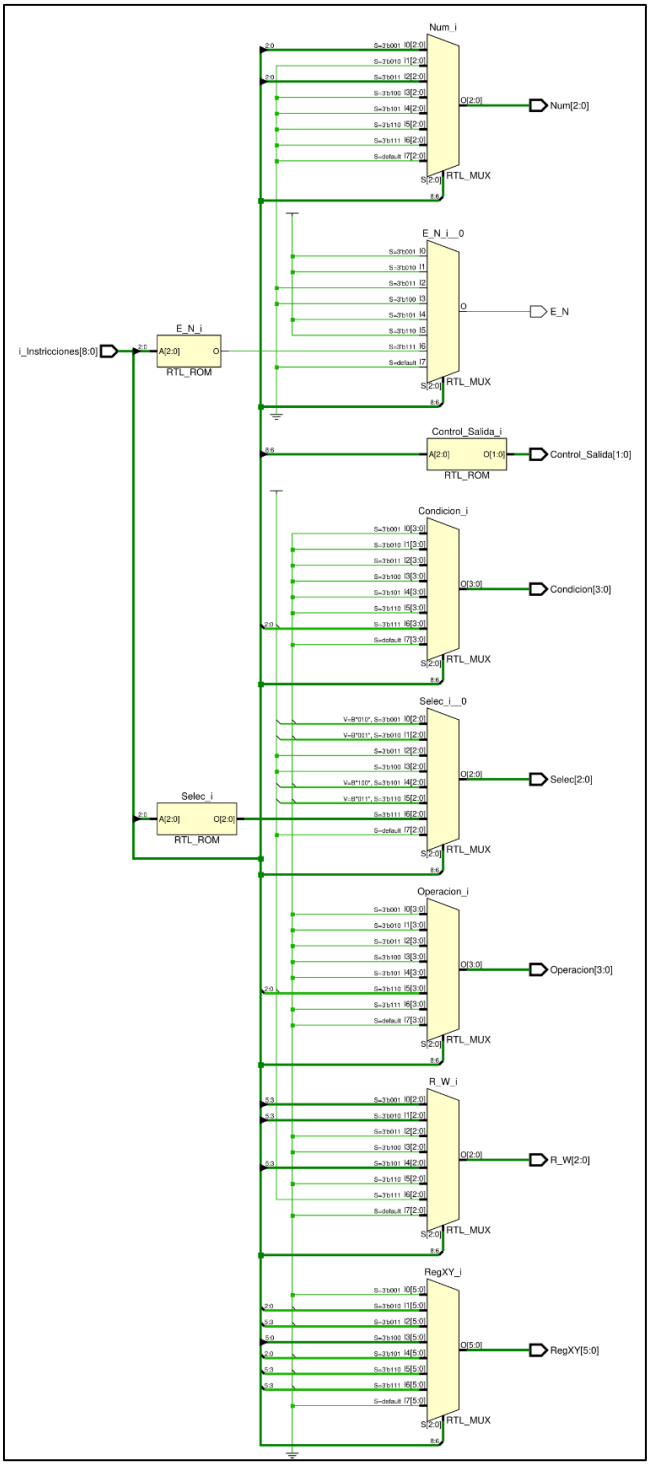


Simulación 1. Microcontrolador de 8 Bits.

Fecha de Entrega: 18 noviembre 2020

Unidad de Control

El esquema 2 muestra el diagrama que nos proporciona vivado en el apartado de esquemático, en el cual se logra apreciar el bloque correspondiente a la unidad de control, así como sus señales de entrada y salida.



Esquema 2. Unidad de Control.

La Unidad de control que diseñamos recibe una única entrada a través de un bus de 9 bits, para probar su funcionamiento simulamos 8 diferentes combinaciones para dicho bus donde cada combinación realiza una de las instrucciones asignadas en nuestro set de instrucciones. La instrucción a ralizar está determinada por los 3 bit más significativos del bus i_instrucciones_tb.

La primera instrucción simulada fue la no operación. Asignándo el valor de 000 en el bus de i_instrucciones_tb obtenemos en las salidas valores de 0 excepto en la salida Selec_tb donde obtenemos un 7 ya que así lo establecimos como valor por defecto para cuando se realice esta instrucción y todas aquellas donde no sea necesario escribir en el banco de registros.

La segunda instrucción simulada con la instrucción 054 representa una carga de un numero en un registro, es por ello que en la señal Num_tb se muestra un 4 ya que este es el número a cargar mientras que en la señal R_W aparece la dirección del registro, la señal E_N_tb cambia su valor a 1 para habilitar la escritura en los registros y en la señal Selec_tb cambia a 2.

La instrucción 09d es una carga en los registros del dato que llegue a través del bus de entrada de datos. Con R_W_tb se selecciona el registro 3 donde se va a cargar el dato, Selec_tb cambia a 1, Control_Salida_tb pasa de 0 a 1 ya que esta se encargará de controlar al bloque control de bus. La señal RegXY_tb muestra las direcciones de los registros que se van a estar leyendo siendo en este caso el registro 0 y el registro 5.

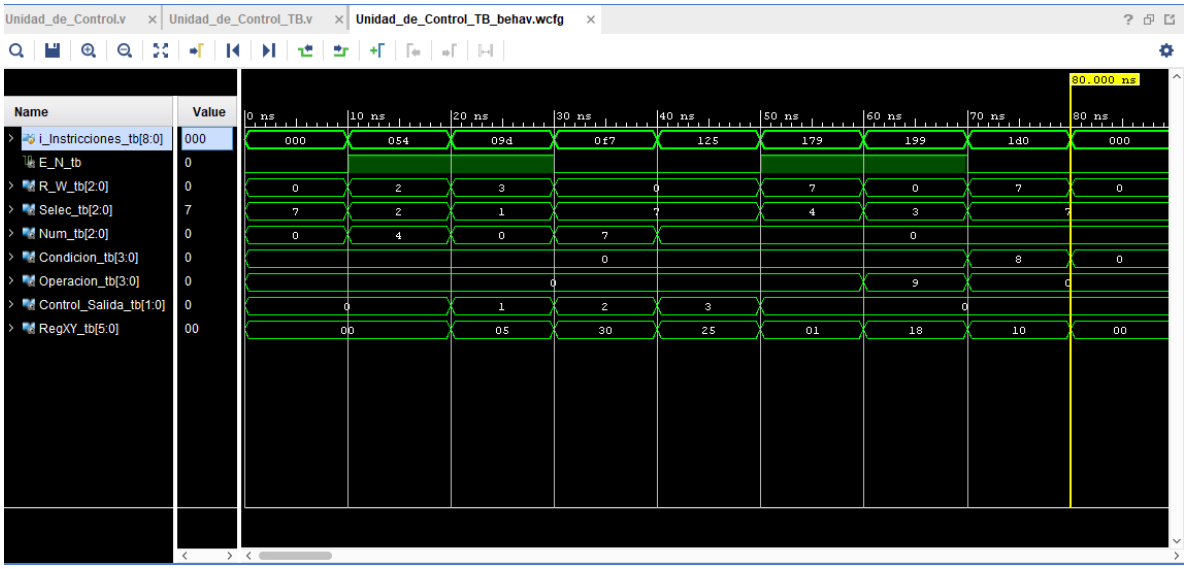
La instrucción 0f7 es una instrucción de almacenamiento de un numero en la dirección de memoria seleccionada a través de los 3 bit más significativos de RegXY_tb, observamos que el número es un 7 y el registro seleccionado es el 6.

Con la instrucción 125 tenemos un almacenamiento del contenido de un registro en la dirección de memoria determinada por otro registro, la señal RegXY_tb indica la dirección de los registros a leer, para este caso los registros 4 y 5. Con la señal Control_Salida_tb controlando a al Control_de_Bus se determinará las salidas hacia las memorias.

La siguiente instrucción asignada con el valor de 179 es la instrucción para mover los datos de un registro a otro, bit más significativo de RegXY_tb son la dirección del registro a mover y R_W_tb es la dirección del registro donde se va a escribir. Entonces tenemos que se va a mover el contenido del registro 1 al registro 7.

Para simular una operación de resta designamos la instrucción con el valor de 199. En la señal de Operacion_tb notamos un valor de 9 el cual corresponde a la resta. En RegXY_tb aparece seleccionadas las direcciones de los registros 0 y 3 que serán los operandos del ALU.

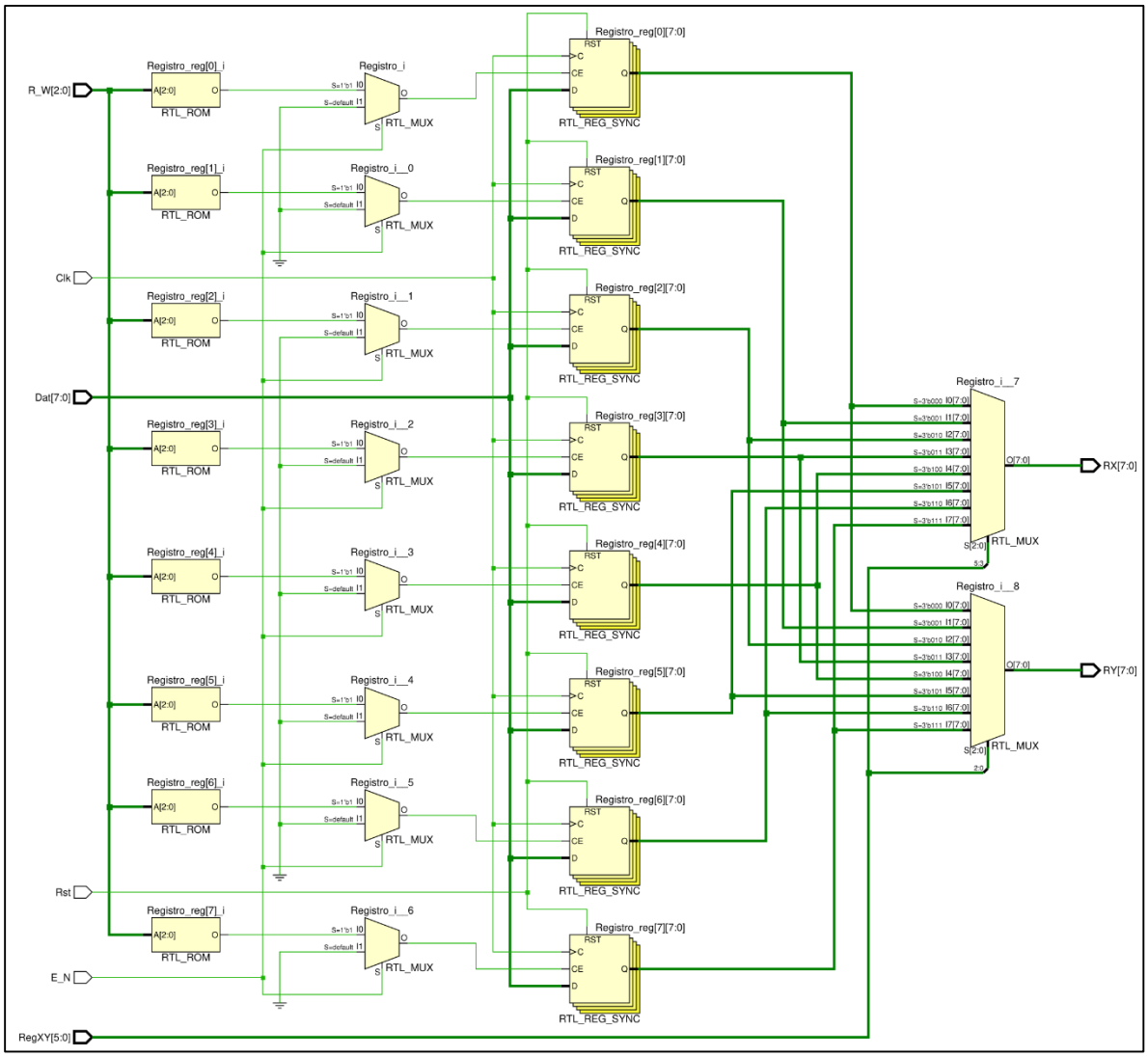
Finalmente, la instrucción 1d0 representa un salto condicional. La condición seleccionada es la 0 aunque se presenta como 8 debido al bit que usaremos como habilitador de la Unidad_de_Salto. En RegXY_tb se muestra la dirección del registro 2 ya que es donde la instrucción indica que se debe saltar al PC.



Simulación 2. Unidad de Control.

Banco de Registros

El esquema 3 muestra el diagrama que nos proporciona vivado en el apartado de esquemático, en el cual se logra apreciar el bloque correspondiente al banco de registro, así como sus señales de entrada y salida.



Esquema 3. Banco de Registros.

En el banco de registros cada ciclo de reloj se estará realizando una acción, ya sea almacenar un dato o simplemente leer lo que hay en sus registros.

Durante el primer ciclo de reloj el reset está activo y a las entradas aún no se les asigna un valor.

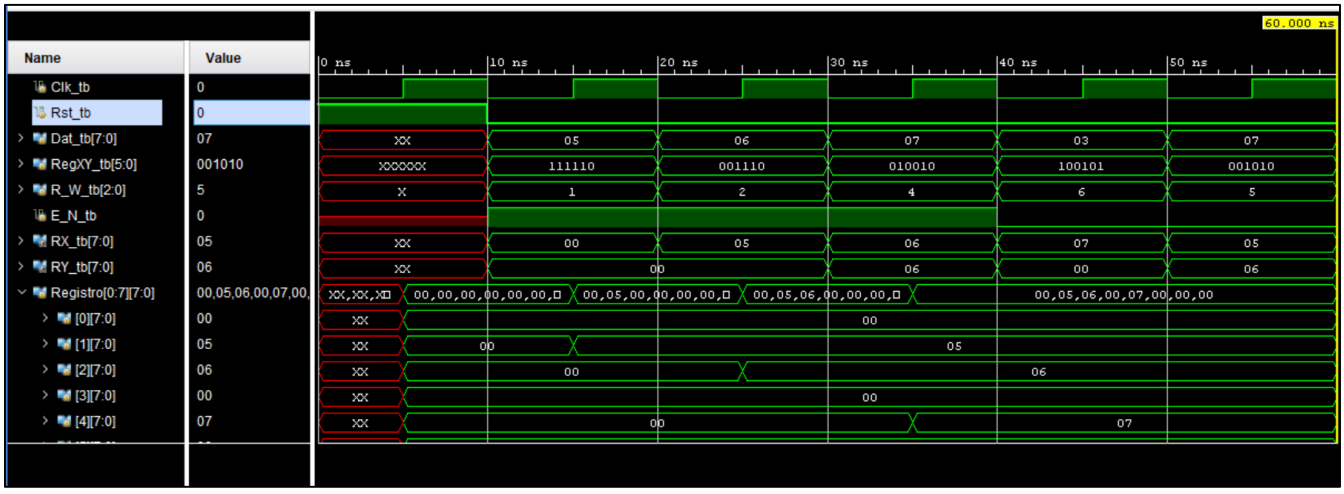
Durante el segundo ciclo el reset se desactiva. En la entrada Dat_tb tenemos un valor de 05 y como la señal E_N_tb está en 1 significa que guardaremos ese dato en el registro 1 ya que es el que indica la señal R_W_tb. En dato se guarda en los registros durante el flanco positivo de reloj y se muestra en la posición 2 de la señal Registros. La entrada RegXY_tb determina los registros que se mostraran en las salidas RX_tb y RY_tb, estos son los registros 7 y 6 pero al contener como dato 0 las salidas muestran este mismo valor.

Durante el tercer ciclo de reloj el dato en Dat_tb cambia a 06 y se guarda durante el flanco positivo de reloj en el registro 2. En la salida RX_tb sale el dato contenido en el registro 1 el cual es 05 y en RY_tb el dato en el registro 6 que es 0.

Durante el cuarto ciclo de reloj se guarda el dato 07 en el registro 4. En la salida RX_tb y en la salida RY_tb se tiene el dato 06 guardado originalmente en el registro 2.

Para el quinto ciclo de reloj la señal E_N_tb pasa de 1 a 0 por lo que independientemente del dato que se encuentre en Dat_tb y de la dirección que indique R_W_tb no se guardara nada en los registros. De la entrada RegXY_tb tenemos la dirección de los registros 4 y 5 cuyos datos será leído a través de las salidas RX_tb y RY_tb respectivamente.

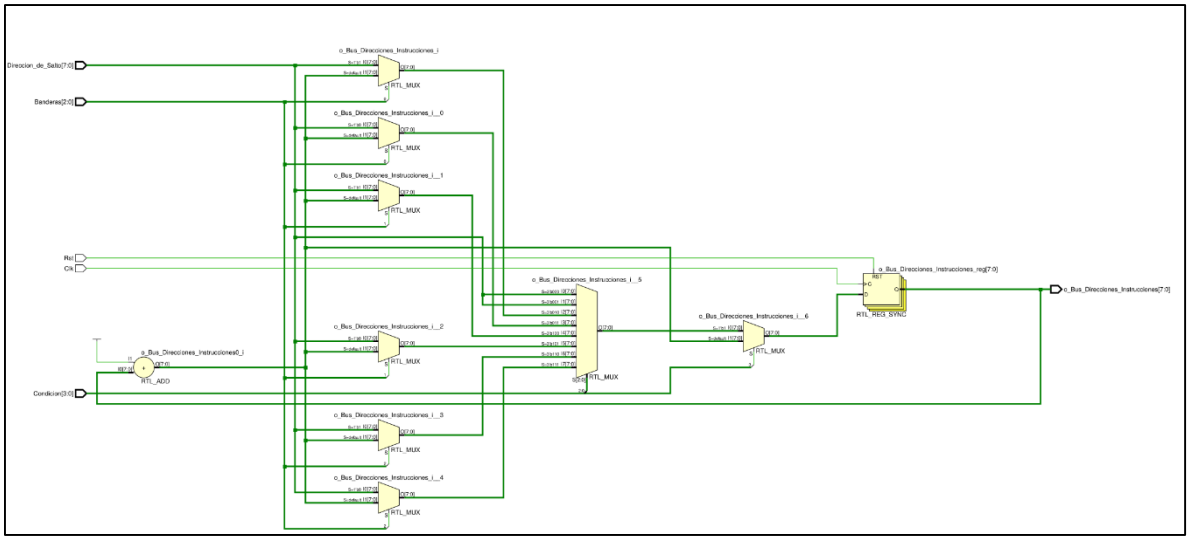
En el sexto ciclo de reloj tampoco se guarda ningún dato en los registros y solo se leen los datos de los registros 1 y 2 por lo que en RX_tb se muestra 05 y en RY_tb aparece un 06.



Simulación 3. Banco de Registros.

Unidad de Saltos

El esquema 4 muestra el diagrama que nos proporciona vivado en el apartado de esquemático, en el cual se logra apreciar el bloque correspondiente a la unidad de saltos, así como sus señales de entrada y salida.



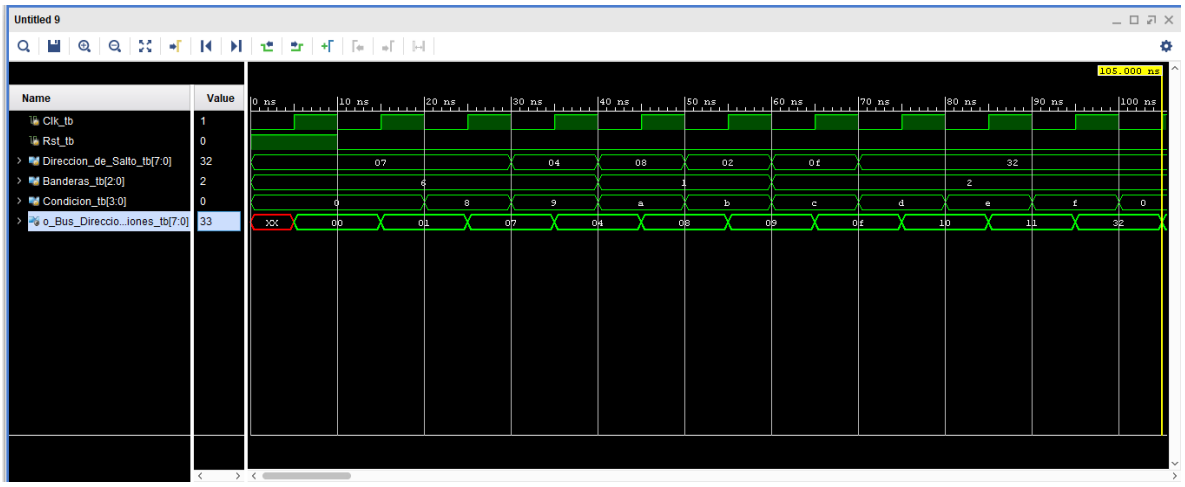
Esquema 4. Unidad de Saltos.

En la simulación de la unidad de saltos probamos las 8 posibles condiciones asignando primero a la señal Condicion_tb un valor de 0 para que no se evalué ninguna de las condiciones. Observamos que la salida incrementa en 1.

Posteriormente probamos la condición 0 que se muestra en 4 bits como 8 debido a que el bit más significativo permanecerá en 1 siempre que evaluemos una condición. En esta condición el valor de la salida o_Bus_Direcion_instrucciones_tb tomara el valor de la señal Direccion_de_Salto lo cual se ve reflejado en el flanco positivo de reloj inmediato. Para Condicion_tb igual a a se comprueba si la señal banderas es igual a 1 (bandera Z verdadera), como la condición se cumple nuevamente la salida toma el valor de Direccion_de_Salto.

Por otra parte, cuando Condicion_tb es igual a b se comprueba justo lo contrario que en la condición anterior, es decir, que banderas no sea igual a 1. Como la condición no se cumple la señal de salida aumenta en 1.

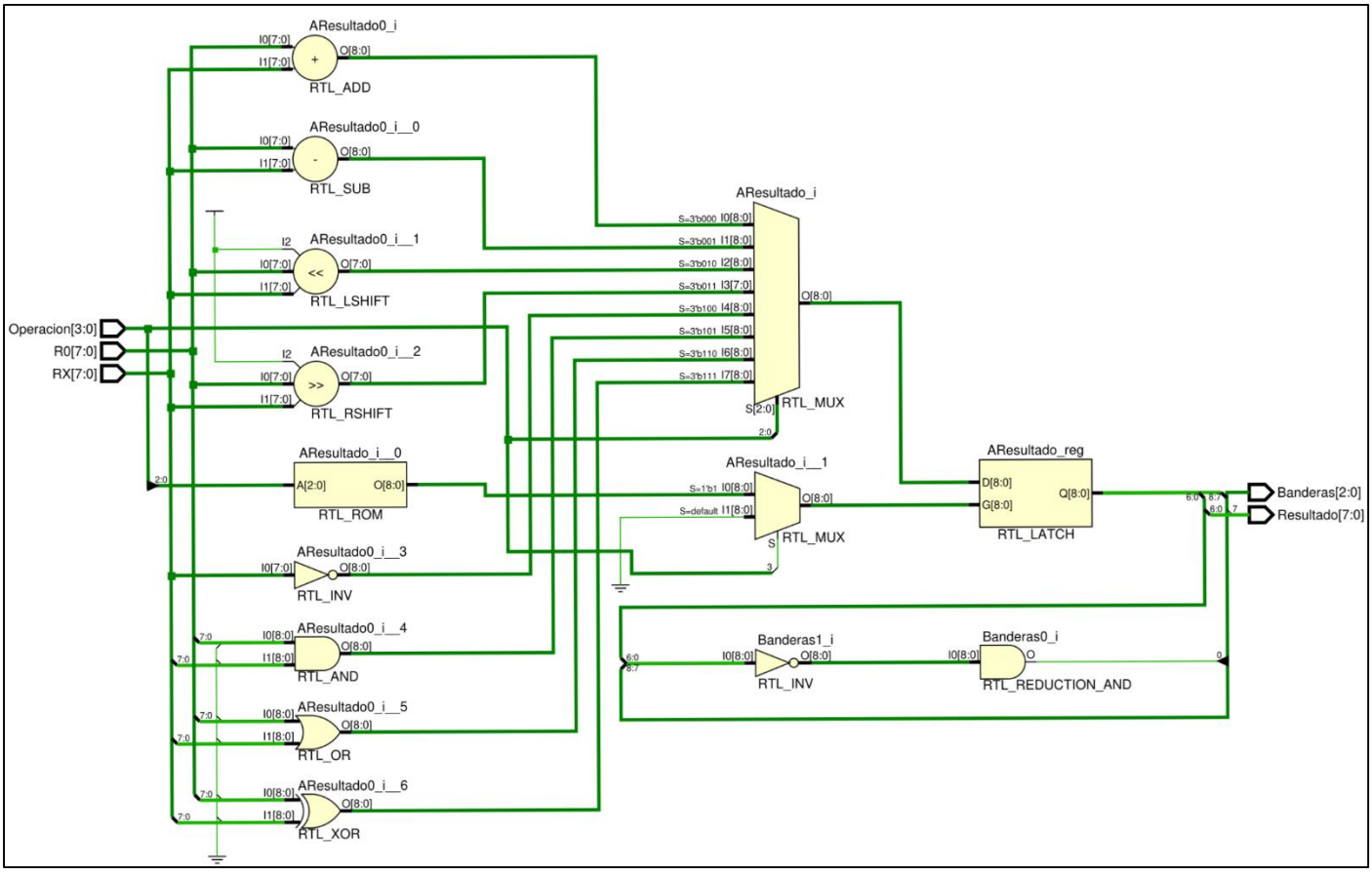
El resto de las condiciones evaluadas a partir de la señal Condicion_tb siguen el mismo patrón que las mencionadas anteriormente con la diferencia de que se comprueban valores distintos para la señal banderas. Cada que la condición se cumple se cambia el valor de la salida a distintos valores de Direccion_de_Salto de lo contrario el valor de la salida o_Bus_Direcion_instrucciones_tb incrementa en 1.



Simulación 4. Simulación de Unidad de Saltos.

ALU

El esquema 5 muestra el diagrama que nos proporciona vivado en el apartado de esquemático, en el cual se logra apreciar el bloque correspondiente a la ALU, así como sus señales de entrada y salida.



Esquema 5. ALU.

De acuerdo a nuestro set de instrucciones la unidad aritmético lógica debe de ser capaz de realizar 8 operaciones distintas de las cuales 2 son operaciones aritméticas, 2 de desplazamiento y 4 lógicas.

La unidad aritmético lógica o ALU tiene 2 entradas que corresponden cada una a uno de los operandos con los que se va a realizar una operación, una tercera entrada permite controlar si se va a realizar o no una operación, en caso de que se vaya a realizar también indica la operación.

Mediante la presente simulación comprobamos el funcionamiento de cada una de las operaciones que son posible ejecutar en nuestra ALU. Para la simulación utilizamos los mismos valores para cada operando y a partir de estos realizamos una operación distinta cada 10ns. El valor establecido para un operando está determinado en la señal R0_tb como 05 y el segundo operando está establecido en RX_tb con un valor de 03. Como primer valor para la señal de entrada Operación_tb tenemos 0, es decir, no se va a realizar ninguna operación por lo tanto tenemos un resultado de 0.

La segunda operación a realizar es la operación 0 del set de instrucciones la cual representa una suma, el resultado de sumar los operandos se muestra en la salida Resultados_tb, el resultado es correcto ya que muestra un valor de 08. La salida Bandera_tb vale 0 dabido a que el resultado no presenta ningún acarre, tampoco se trata de un numero negativo o un cero.

La operación 1 establecida en la entrada Operación_tb como 8 es una resta entre 05 y 03, el resultado es 02 por lo que es correcto. Nuevamente no se activa ninguna bandera.

La operación 2 indicada como a es un desplazamiento a la izquierda del valor 05 de 03 posiciones. El resultado es 28.

La operación 3 indicada como b es un desplazamiento de 03 posiciones a la derecha del valor 05. Como los bits menos significativos se desplazan a la derecha quedan únicamente ceros por lo que la bandera cero se activa.

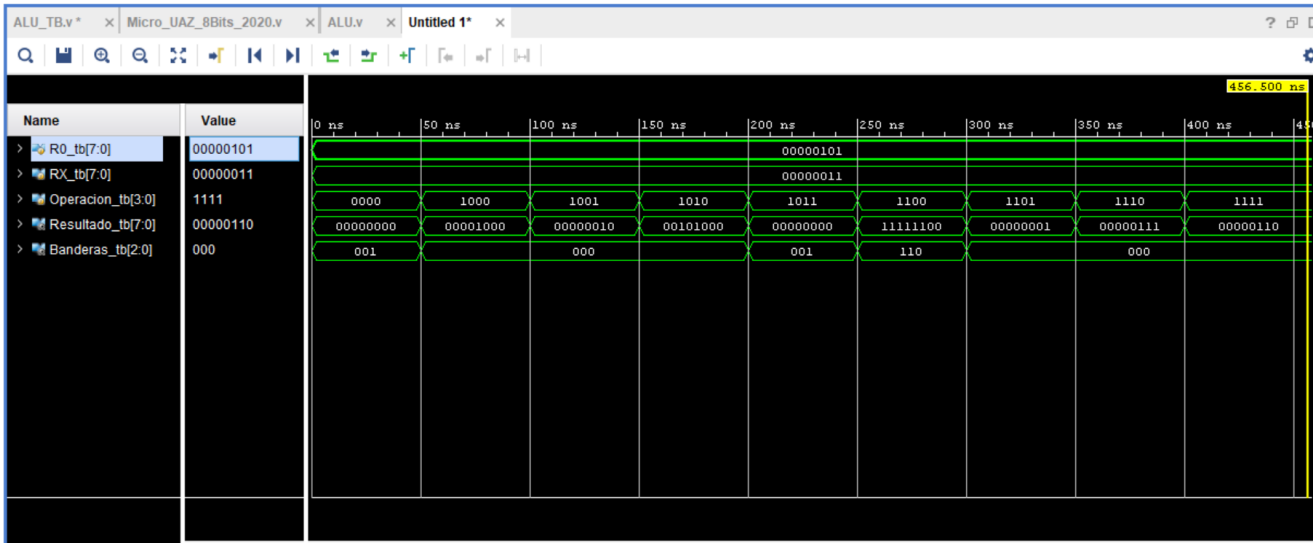
La operación 4 correspondiente al valor c de Opercion_tb es la operación lógica de negación o NOT del valor 03, al negar cada bit obtenemos un resultado de fc, se produce un

acarreo y el dato cambia de positivo a negativo por lo que se activan las banderas correspondientes.

La quinta operación definida con el valor d corresponde a la AND, el resultado es 01 por lo que tampoco se activa una bandera.

La sexta operación definida por el valor e es un OR entre los operandos, el resultado es de 07 y tampoco se activa ninguna bandera.

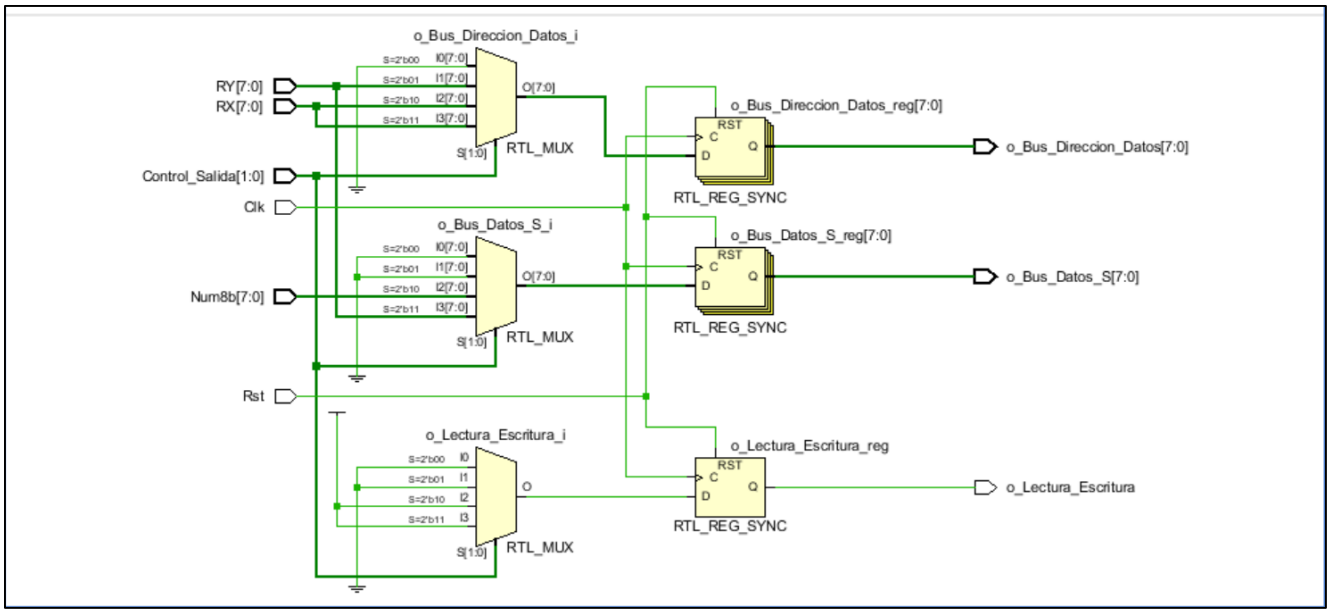
Finalmente, la operación 7 es un XOR entre 05 y 03 que como resultado da 06 sin activar ninguna bandera.



Simulación 5. ALU.

Control de Bus

El esquema 6 muestra el diagrama que nos proporciona vivado en el apartado de esquemático, en el cual se logra apreciar el bloque correspondiente al control de bus, así como sus señales de entrada y salida.



Esquema 6. Control de Bus.

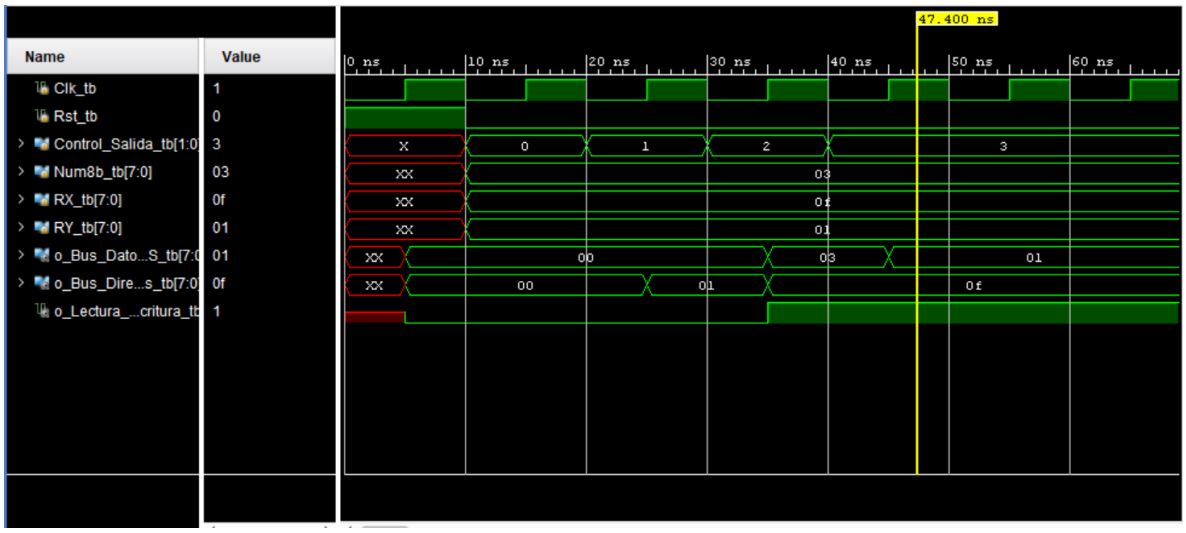
En la siguiente simulación mostramos el funcionamiento de la unidad de control. Aparte de las señales de reloj y reset, este bloque tiene 4 señales de entrada de las cuales 3 contendrán datos que serán trasferidos a las salidas mientras que la señal de entrada Control_Salida_tb controlara el destino de los datos de entrada.

Cuando la señal Control_Salida_tb vale 0 significa que en las 3 salidas se mantendrán en 0.

Cuando la señal Control_Salida_tb vale 1 el dato en la entrada RY_tb saldrá a través de la salida o_Bus_Direccion_Datos mientras que las demás salidas conservan un valor de 0.

Al cambiar la señal Control_Salida_tb a 2 el dato en la entrada Num8b_tb saldrá por la salida o_Bus_Datos_S mientras que el dato en Rx_tb lo obtendremos en la salida o_Bus_Direccion_Datos. Ahora la señal o_Lectura_Escritura pasa de 0 a 1.

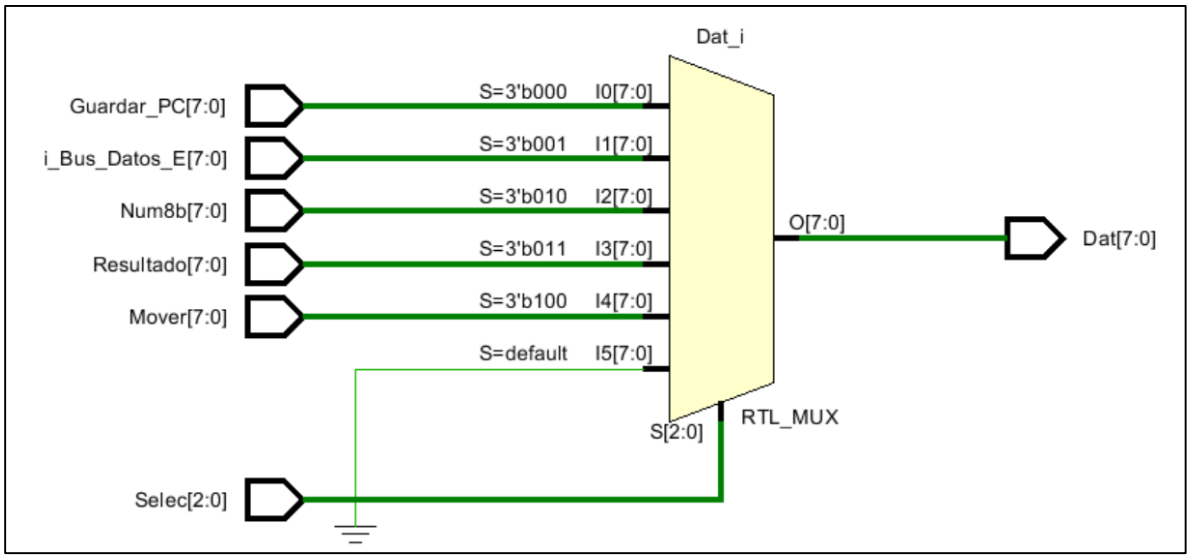
Finalmente cuando Control_Salida_tb es igual a 3 tendremos en la salida o_Bus_Datos_S el dato en RY_tb y en la salida o_Bus_Direccion_Datos al dato de la entrada RX_tb, la señal o_Lectura_Escritura toma valor de 1.



Simulación 6. Control de Bus.

Mux1

El esquema 7 muestra el diagrama que nos proporciona vivado en el apartado de esquemático, en el cual se logra apreciar el bloque correspondiente al mux1, así como sus señales de entrada y salida.



Esquema 7. MUx1.

En la simulación correspondiente al multiplexor se realizaron distintas pruebas para comprobar el correcto funcionamiento del multiplexor el cual direccionar una sola entrada hacia la salida ya que tendremos más de una entrada y solo una sola salida. esto se lleva acabó gracias al selector de entradas.

Por lo que en nuestro caso contamos con 6 entradas, las cuales 5 corresponden a datos (Guardar_PC, i_Bus_Datos_E, Num8b, Resultado y Mover) y la sexta entrada pertenece al selector (selec). Por lo que tenemos una sola salida llamada Dat la cual dependerá del selector como ya mencionamos anteriormente.

La primera prueba consistió en darle un valor de 0 al selector (selec), lo que permite direccionar la entrada Guardar_PC (Datos_0) en la salida Dat. Es por eso que se puede visualizar el valor de Guardar_PC en la salida

Para posteriormente modificar el valor de selec a 1, para direccionar ahora la entrada i_Bus_Datos_E (Datos_1) en la salida.

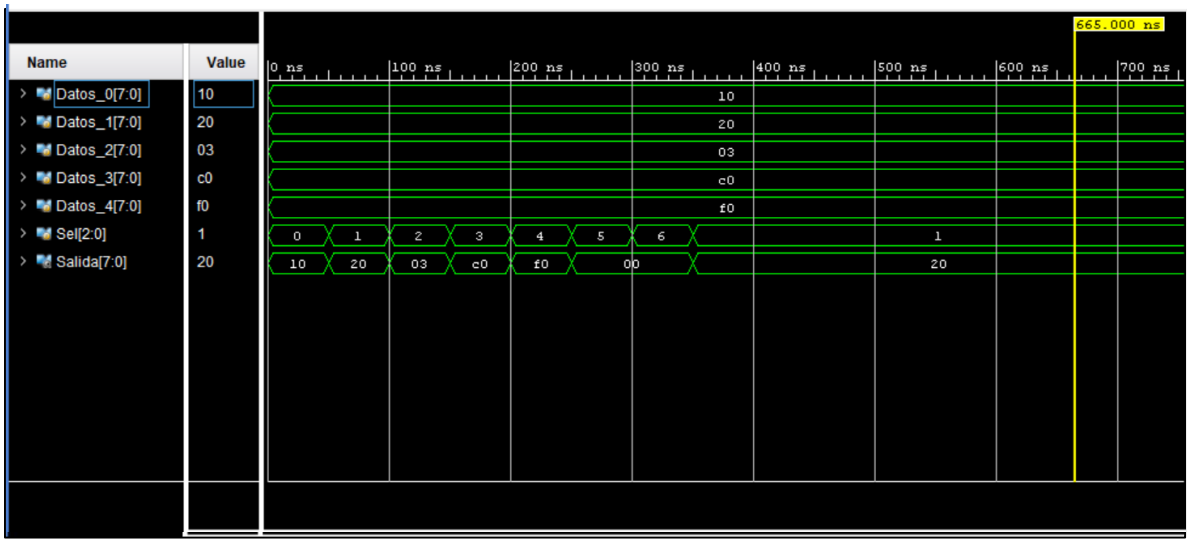
Luego volvimos a remplazar el valor de selec por un 2 lo que provoca un direccionamiento de la entrada Num8b (Datos_2) hacia la salida.

En la cuarta prueba se actualizo el valor del selector a 3, lo que provoco que la salida tomara el valor de la entrada Resultado (Datos_3).

En la prueba 5 la entrada selec toma el valor de 4 por lo que se ve reflejado en la salida Dat el valor de la entrada Mover(Datos_4).

En este punto ya no tenemos entradas que direccionar y el selector está definido con un tamaño de 3 bits, es así que tenemos 2^3 combinaciones o datos a seleccionar es por eso que al elaborar el mux declaramos un default donde entran las combinaciones sobrantes. Por lo que en la simulación es posible visualizar que al colocarle un valor de 5 y 6 al sector la salida es 0 ya que así está definido en el default de nuestra descripción del bloque mux.

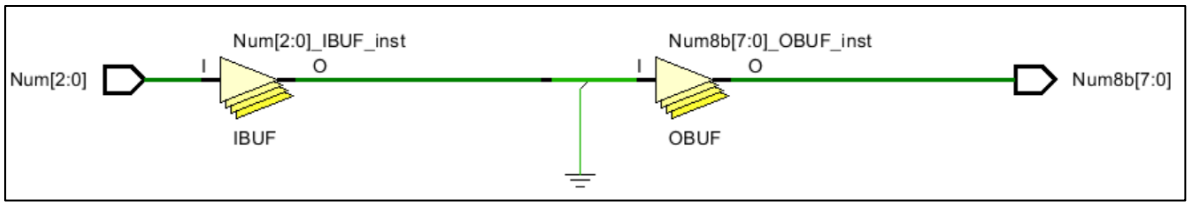
Por ultimo volvimos a colocar un valor de 1 al selector por lo que la salida Dat vuelve a tomar el valor de la entrada i_Bus_Datos_E.



Simulación 7. Mux1.

ConNum

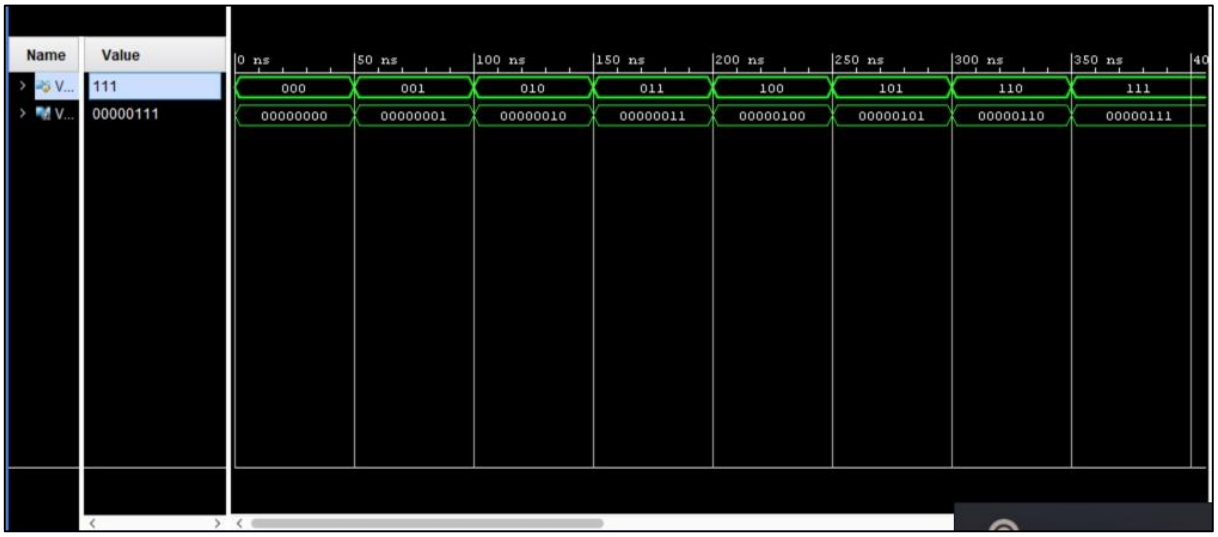
El esquema 8 muestra el diagrama que nos proporciona vivado en el apartado de esquemático, en el cual se logra apreciar el bloque correspondiente al ConNum, así como sus señales de entrada y salida.



Esquema 8. ConNum.

El bloque ConNum es muy sencillo y fácil de comprender ya que su tarea es agregar o concatenar seis 0 a la señal Num de 3 bits con la finalidad de convertir dicha señal en una de 8 bits.

Este bloque solo tiene una señal de entrada de 3 bits que se llama VNum y una salida de 8 bits llamada VNum8b. Por lo en la simulación se ingresaron valores del 0-7 en la entrada para comprobar que en la salida se pueda visualizar el mismo número, pero ahora convertido a 8 bits.



Simulación 8. ConNum.

Conclusión

Con el desarrollo del presente proyecto aprendimos a diseñar nuestro propio microcontrolador a partir del set de instrucciones específico. Para ello fue necesario comprender el funcionamiento de los microprocesadores, de la manera en que se integran dentro de una computadora y de sus capacidades. Dado que nuestro microprocesador es de arquitectura tipo Harvard y con un set de instrucciones tipo RISC fue necesario aprender las diferencias que hay entre las distintas arquitecturas y tipos de set de instrucciones. Aprendimos como es la manera en que se lleva a cabo la ejecución de instrucciones en los microprocesadores, los modos de direccionamiento de datos, arquitecturas, así mismo nos fue de utilidad para reforzar aquellos conocimientos anteriormente adquiridos en la materia de electrónica digital, principalmente en la elaboración de registros, maquias de estado, decodificadores, entre otros que nosotros como estudiantes no dominábamos todavía.

Comenzar nuestro procesador a partir de un set de instrucciones especifico facilito mucho la elaboración del microprocesador, ya que elaborar un set de instrucciones, es un trabajo que requiere mucho tiempo y planeación, otro factor importante que nos fue de utilidad fue aplicar la metodología top down ya que de esta manera nos asegurarnos de tener los componentes necesarios para cumplir con nuestro set de instrucciones a través de prueba y error, pudimos identificar soluciones a los problemas que presentaban nuestros diseños iniciales y así una vez terminado el diseño general del microcontrolador fuera más sencillo realizar las descripciones funcionales de cada componente y sobre todo facilitar la codificación.

Referencias

[1] Ecured.cu. 2020. *Microcontrolador - Ecured*. [online] Available at: <https://www.ecured.cu/Microcontrolador> [Accessed 18 November 2020].

[2] "Estructura de computadores", *Cv.uoc.edu*, 2020. [Online]. Available: http://cv.uoc.edu/annotation/8255a8c320f60c2bfd6c9f2ce11b2e7f/619469/PID_00218274/PID_00218274.html#w31aab5c13. [Accessed: 18- Nov- 2020].

[3] "1.3 Principios de Arquitectura RISC", *Cidecame.uaeh.edu.mx*, 2020. [Online]. Available: http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro20/13_principios_de_arquitectura_risc.html. [Accessed: 18- Nov- 2020].

[4] "Sistema digital y sistema analógico: diferencias, ventajas y desventajas", *Diferenciador*, 2020. [Online]. Available: https://www.diferenciador.com/sistema-digital-y-sistema-analogico/. [Accessed: 18- Nov- 2020].

[5] "Google Traductor", *Translate.google.com*, 2020. [Online]. Available: https://translate.google.com/translate?hl=es-419&sl=en&u=https://www.prowesscorp.com/what-is-fpga/&prev=search&pto=aue. [Accessed: 18- Nov- 2020].

[6] ".::VERILOG - TUTORIAL VERILOG::.", *Matpic.com*, 2020. [Online]. Available: http://www.matpic.com/esp/vhdl/verilog.html. [Accessed: 18- Nov- 2020].

[7] "Combinational and Sequential Circuits - GeeksforGeeks", *GeeksforGeeks*, 2020. [Online]. Available: https://www.geeksforgeeks.org/combinational-and-sequential-circuits/. [Accessed: 18- Nov- 2020].

Apéndice A Códigos en Verilog

Micro_UAZ_8Bits_2020

```
1   module Micro_UAZ_8Bits_2020(
2       input Clk,
3       input Rst,
4       input [8:0] i_Instrucciones,
5       input [7:0] i_Bus_Datos_E,
6       output [7:0] o_Bus_Datos_S,
7       output [7:0] o_Bus_Direccion_Datos,
8       output  o_Lectura_Escritura,
9       output [7:0] o_Bus_Direcciones_Instrucciones
10      );
11
12      wire [1:0] Control_Salida;
13      wire [2:0] Num;
14      wire [2:0] Selec;
15      wire [5:0] RegXY;
16      wire [2:0]R_W;
17      wire E_N;
18      wire [3:0] Operacion;
19      wire [3:0] Condicion;
20      wire [7:0] Num8b;
21      wire [7:0] Resultado;
22      wire [7:0] Dat;
23      wire [7:0] RX;
24      wire [7:0] RY;
25      wire [2:0] Banderas;
26
27      Unidad_de_Control Bloque_1(
28          .i_Instricciones(i_Instricciones),
29          .Control_Salida(Control_Salida),
30          .Num(Num),
31          .Selec(Selec),
32          .RegXY(RegXY),
33          .R_W(R_W),
34          .E_N(E_N),
```

```
35          .Operacion(Operacion),
36          .Condicion(Condicion)
37      );
38
39      ConNum Bloque_2 (
40          .Num(Num),
41          .Num8b(Num8b)
42      );
43
44      Mux1 Bloque_3(
45          .Resultado(Resultado),
46          .Num8b(Num8b),
47          .i_Bus_Datos_E(i_Bus_Datos_E),
48          .Guardar_PC( o_Bus_Direcciones_Instrucciones),
49          .Mover(RY),
50          .Selec(Selec),
51          .Dat(Dat)
52      );
53
54      Banco_de_Registros Bloque_4 (
55          .Clk(Clk),
56          .Rst(Rst),
57          .Dat(Dat),
58          .RegXY(RegXY),
59          .R_W(R_W),
60          .E_N(E_N),
61          .RX(RX),
62          .RY(RY),
63      );
64
65      ALU Bloque_5 (
66          .R0(RY),
67          .RX(RX),
68          .Operacion(Operacion),
69          .Resultado(Resultado),
70          .Banderas(Banderas)
71      );
72
73      Unidad_de_Saltos Bloque_6(
```

```
74          .Clk(Clk),
75          .Rst(Rst),
76          .Direccion_de_Salto(RX),
77          .Banderas(Banderas),
78          .Condicion(Condicion),
79          .o_Bus_Direcciones_Instrucciones(o_Bus_Direcciones_Instrucciones)
80      );
81
82      Control_de_Bus Bloque_7 (
83          .Clk(Clk),
84          .Rst(Rst),
85          .Num8b(Num8b),
86          .Control_Salida(Control_Salida),
87          .RY(RY),
88          .RX(RX),
89          .o_Bus_Datos_S(o_Bus_Datos_S),
90          .o_Bus_Direccion_Datos(o_Bus_Direccion_Datos),
91          .o_Lectura_Escritura(o_Lectura_Escritura)
92      );
93
94  endmodule
```

Código 1. Microcontrolador de 8 Bits.

Unidad_de_Control

```
1   module Unidad_de_Control(
2           input [8:0] i_Instricciones,
3           output reg E_N,
4           output reg [2:0] R_W,
5           output reg [2:0] Selec,
6           output reg [2:0] Num,
7           output reg [2:0] Condicion,
8           output reg [3:0] Operacion,
9           output reg [1:0] Control_Salida,
10          output reg [5:0] RegXY
11  );
12
13
14  always @(i_Instricciones)begin
15                  case(i_Instricciones[8:6] ) //Evalua la instruccion a realizar
16                          3'b001: begin // LOAD Num
17                                  E_N <= 1;
18                                  R_W <= i_Instricciones[5:3];
19                                  Selec <= 3'b010;
20                                  Num <= i_Instricciones[2:0];
21                                  Condicion <= 4'b0000;
22                                  Operacion <= 4'b0000;
23                                  Control_Salida <= 2'b00;
24                                  RegXY <= 6'b000000;
25                          end
26                          3'b010: begin //LOAD EN RX
27                                  E_N <= 1;
28                                  R_W <= i_Instricciones[5:3];
29                                  Selec <= 3'b001;
30                                  Num <= 3'b000;
31                                  Condicion <= 4'b0000;
32                                  Operacion <= 4'b0000;
33                                  Control_Salida <= 2'b01;
34                                  RegXY[5:3] <= 3'b000;
35                                  RegXY[2:0] <= i_Instricciones[2:0];
36                          end
37                          3'b011: begin // STORE Num
38                                  E_N <= 0;
39                                  R_W <= 3'b000;
40                                  Selec <= 3'b111;
41                                  Num <= i_Instricciones[2:0];
42                                  Condicion <= 4'b0000;
```

```
43                          Operacion <= 4'b0000;
44                          Control_Salida <= 2'b10;
45                          RegXY[5:3] <= i_Instricciones[5:3];
46                          RegXY[2:0] <= 3'b000;
47                      end
48                      3'b100: begin //STORE [RY] en RX
49                          E_N <= 0;
50                          R_W <= 3'b000;
51                          Selec <= 3'b111;
52                          Num <= 3'b000;
53                          Condicion <= 4'b0000;
54                          Operacion <= 4'b0000;
55                          Control_Salida <= 2'b11;
56                          RegXY <= i_Instricciones[5:0];
57                      end
58                      3'b101: begin // Move
59                          E_N <= 1;
60                          R_W <= i_Instricciones[5:3];
61                          Selec <= 3'b100;
62                          Num <= 3'b000;
63                          Condicion <= 4'b0000;
64                          Operacion <= 4'b0000;
65                          Control_Salida <= 2'b00;
66                          RegXY[5:3] <= 3'b000;
67                          RegXY[2:0] <= i_Instricciones[2:0];
68                      end
69                      3'b110: begin //Math
70                          E_N <= 1;
71                          R_W <= 3'b000;
72                          Selec <= 3'b011;
73                          Num <= 3'b000;
74                          Condicion <= 4'b0000;
75                          Operacion[2:0] <= i_Instricciones[2:0];
76                          Operacion[3] <= 1;
77                          Control_Salida <= 2'b00;
78                          RegXY[5:3] <= i_Instricciones[5:3];
79                          RegXY[2:0] <= 3'b000;
80                      end
81                      3'b111: begin  //Jump
82                          //E_N <= 0;
83                          R_W <= 3'b111;
84                          Num <= 3'b000;
85                          Condicion[2:0] <= i_Instricciones[2:0];
86                          Condicion[3] <= 1;
87                          Operacion <= 4'b0000;
```

```
88                                      Control_Salida <= 2'b00;
89                                      RegXY[5:3] <= i_Instricciones[5:3];
90                                      RegXY[2:0] <= 3'b000;
91                                          if(i_Instricciones[2:0]==2'b001)begin
92                                              E_N <= 1;
93                                              Selec <= 0;
94                                          end
95                                          else begin
96                                              E_N <= 0;
97                                              Selec <= 7;
98                                          end
99                                  end
100                                 default begin  //NO Operacion
101                                     E_N <= 0;
102                                     R_W <= 3'b000;
103                                     Selec <= 3'b111;
104                                     Num <= 3'b000;
105                                     Condicion <= 4'b0000;
106                                     Operacion <= 4'b0000;
107                                     Control_Salida <= 2'b00;
108                                     RegXY <= 6'b000000;
109                                 end
110                         endcase
111                 end
112 endmodule
```

Código 2. Unidad de Control.

Banco_de_Registros

.

```
1   module Banco_de_Registros (
2           input Clk,
3           input Rst,
4           input [7:0] Dat,
5           input [5:0] RegXY,
6           input [2:0] R_W,
7           input E_N,
8           output [7:0] RX,
9           output [7:0] RY
10  );
11
12  reg [7:0] Registro[0:7];
13
14          always @(posedge Clk)
15          begin
16                  if(Rst) begin
17                          Registro[0] <= 0;
18                          Registro[1] <= 0;
19                          Registro[2] <= 0;
20                          Registro[3] <= 0;
21                          Registro[4] <= 0;
22                          Registro[5] <= 0;
23                          Registro[6] <= 0;
24                          Registro[7] <= 0;
25
26                  end
27                  else if(E_N)
28                          Registro[R_W] <= Dat;
29          end
30          assign RX = Registro[RegXY[5:3]];
31          assign RY = Registro[RegXY[2:0]];
32  endmodule
```

Código 3. Banco de Registros

ALU

```
1   module ALU(
2   input [7:0] RX,
3   input [7:0] R0,
4   input [3:0] Operacion,
5   output [7:0] Resultado,
6   output [2:0] Banderas);
7
8   reg [8:0] AResultado =0;
9
10  always@(RX,Operacion)
11  begin
12      if (Operacion[3] == 1) begin
13          case (Operacion[2:0])
14              3'b000: begin
15                  AResultado <= R0 + RX;
16              end
17              3'b001: begin
18                  AResultado <= R0 - RX;
19              end
20              3'b010: begin
21                  AResultado[7:0] <= R0 << RX;
22              end
23              3'b011: begin
24                  AResultado <= R0 >> RX;
25              end
26              3'b100: begin
```

```
27              AResultado <= ~ RX;
28          end
29          3'b101: begin
30              AResultado <= R0 & RX;
31          end
32          3'b110: begin
33              AResultado <= R0 | RX;
34          end
35          3'b111: begin
36              AResultado <= R0 ^ RX;
37          end
38      endcase
39    end
40    else begin
41      AResultado <= AResultado;
42    end
43  end
44    assign Resultado = AResultado[7:0];
45    assign Banderas[0] = &(~AResultado);
46    assign Banderas[1] = AResultado[8];
47    assign Banderas[2] = AResultado[7];
48  endmodule
```

Código 4. ALU.

Unidad_de_Saltos

```
1   module Unidad_de_Saltos (
2           input Clk,
3           input Rst,
4               input [7:0] Direccion_de_Salto,
5               input [2:0] Banderas,
6           input [2:0] Condicion,
7          output reg[7:0] o_Bus_Direcciones_Instrucciones
8   );
9
10  always @(posedge Clk)
11  begin
12      if(Rst) begin
13        o_Bus_Direcciones_Instrucciones <= 0;
14     end
15     else begin
16       if (Condicion[3] == 1) begin
17           case (Condicion[2:0])
18              3'b000: begin
19                 o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
20              end
21              3'b001: begin
22                 o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
23              end
24              3'b010: begin //Z
25                    if(Banderas[0])
26                        o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
27                    else
28                        o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
29                    end
30              3'b011: begin //~Z
31                    if(~Banderas[0])
32                        o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
33                    else
34                        o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
35                    end
36              3'b100: begin //C
37                    if(Banderas[1])
38                        o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
39                    else
40                        o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
41                    end
42              3'b101: begin //~C
43                    if(~Banderas[1])
44                        o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
45                    else
46                        o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
47                    end
48              3'b110: begin //N
49                    if(Banderas[2])
50                        o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
51                    else
52                        o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
53                    end
```

```
54                  3'b111: begin //~N
55                      if(~Banderas[2])
56                          o_Bus_Direcciones_Instrucciones <= Direccion_de_Salto;
57                      else
58                          o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
59                      end
60              endcase
61          end
62          else begin
63              o_Bus_Direcciones_Instrucciones <= o_Bus_Direcciones_Instrucciones + 1;
64          end
65      end
66  end
67  endmodule
```

Código 5. Unidad de Saltos.

Control_de_Bus

```
1   module Control_de_Bus (
2   input Clk,
3   input Rst,
4   input [1:0] Control_Salida,
5   input [7:0] Num8b,
6   input [7:0] RX,
7   input [7:0] RY,
8   output reg[7:0] o_Bus_Datos_S,
9   output reg[7:0] o_Bus_Direccion_Datos,
10  output reg o_Lectura_Escritura);
11
12  always@(posedge Clk)
13  begin
14  if (Rst)
15  begin
16  o_Bus_Datos_S <= 0;
17  o_Bus_Direccion_Datos <= 0;
18  o_Lectura_Escritura <= 0;
19  end
20      case (Control_Salida)
21          2'b00: begin
22              o_Bus_Datos_S <= 0;
23              o_Bus_Direccion_Datos <= 0;
24              o_Lectura_Escritura <= 0;
25          end
26          2'b01: begin// LOAD_Registros
27              o_Bus_Datos_S <= 0;
28              o_Bus_Direccion_Datos <= RY;
29              o_Lectura_Escritura <= 0;
30          end
31          2'b10: begin// STORE_Num
32              o_Bus_Datos_S <= Num8b;
33              o_Bus_Direccion_Datos <= RX;
34              o_Lectura_Escritura <= 1;
35          end
36          2'b11: begin // STORE_Registro
37              o_Bus_Direccion_Datos <= RX;
38              o_Bus_Datos_S <= RY;
39              o_Lectura_Escritura <= 1;
40          end
41          default begin
42              o_Bus_Datos_S <= 0;
```

```
43              o_Bus_Direccion_Datos <= 0;
44              o_Lectura_Escritura <= 0;
45          end
46      endcase;
47  end
48  endmodule
```

Código 6. Control de Bus.

Mux1

```
1   module Mux1 (
2       input [7:0] Resultado,
3       input [7:0] Num8b,
4       input [7:0] i_Bus_Datos_E,
5       input [7:0] Guardar_PC,
6       input [7:0] Mover,
7       input [2:0] Selec,
8       output reg[7:0] Dat);
9
10      always@(Selec,Resultado,Num8b,i_Bus_Datos_E,Guardar_PC,Mover)
11              begin
12                      case (Selec)
13                              3'b000: Dat <= Guardar_PC;
14                              3'b001: Dat <= i_Bus_Datos_E;
15                              3'b010: Dat <= Num8b;
16                              3'b011: Dat <= Resultado;
17                              3'b100: Dat <= Mover;
18                              default : Dat <= 0;
19                      endcase
20      end
21  endmodule
```

Código 7. Mux1.

ConNum

```
1   module ConNum(
2           input [2:0] Num,
3           output [7:0] Num8b;
4           );
5
6       reg [4:0] Extencion= 5'b00000;
7
8       assign Num8b = {Extencion, Num};
9
10      endmodule
```

Código 8. ConNum.

Apéndice B Testbench

Micro_UAZ_8Bits_2020_TB

```
1   module Micro_UAZ_8Bits_2020_TB(
2   );
3       reg Clk_tb;
4       reg Rst_tb;
5       reg [8:0] i_Instricciones_tb;
6       reg [7:0] i_Bus_Datos_E_tb;
7       wire [7:0] o_Bus_Datos_S_tb;
8       wire [7:0] o_Bus_Direccion_Datos_tb;
9       wire o_Lectura_Escritura_tb;
10      wire [7:0] o_Bus_Direcciones_Instrucciones_tb;
11
12      Micro_UAZ_8Bits_2020 DUT(
13          .Clk(Clk_tb),
14          .Rst(Rst_tb),
15          .i_Instricciones(i_Instricciones_tb),
16          .i_Bus_Datos_E(i_Bus_Datos_E_tb),
17          .o_Bus_Datos_S(o_Bus_Datos_S_tb),
18          .o_Bus_Direccion_Datos(o_Bus_Direccion_Datos_tb),
19          .o_Lectura_Escritura(o_Lectura_Escritura_tb),
20          .o_Bus_Direcciones_Instrucciones(o_Bus_Direcciones_Instrucciones_tb)
21      );
22
23      initial
24      begin
25          Clk_tb<=0;
26          Rst_tb<=1;
```

```
27          #10 Rst_tb<=0;
28          i_Instricciones_tb <= 9'b000000001; //NO OPERACION
29          i_Bus_Datos_E_tb <= 8'b00001010; //10 en Bus de Datos de entrada
30          #10;
31          i_Instricciones_tb <= 9'b001011011; //LOAD R3,#3
32          #10;
33          i_Instricciones_tb <= 9'b010000011; //LOAD R0,[R3]
34          #10;
35          i_Instricciones_tb <= 9'b011010100; //STORE #4
36          #10;
37          i_Instricciones_tb <= 9'b100010111; //STORE [R2],R7
38          #10;
39          i_Instricciones_tb <= 9'b101011000; //MOVE R3,R0
40          #10;
41          i_Instricciones_tb <= 9'b110011001; //MATH R3,-
42          #10;
43          i_Instricciones_tb <= 9'b111011010; //JUMP [R3],2
44          #10;
45          i_Instricciones_tb <= 9'b001010111; //LOAD R2,#7
46          #10;
47          i_Instricciones_tb <= 9'b110010000; //MATH R2,+
48          #10;
49          i_Instricciones_tb <= 9'b111010001; //JUMP [R2],1
50          #10;
51
52      end
53      always@(Clk_tb) begin
54          #5 Clk_tb<=~Clk_tb;
55      end
56  endmodule
```

Testbench 1. Micro_UAZ_8Bits_2020.

Unidad_de_Control_TB

```
1   module Unidad_de_Control_TB(
2   );
3           reg [8:0] i_Instricciones_tb;
4           wire E_N_tb;
5       wire [2:0] R_W_tb;
6       wire [2:0] Selec_tb;
7       wire [2:0] Num_tb;
8       wire [3:0] Condicion_tb;
9       wire [3:0] Operacion_tb;
10          wire [1:0] Control_Salida_tb;
11          wire [5:0] RegXY_tb;
12
13          Unidad_de_Control DUT(
14          .i_Instricciones(i_Instricciones_tb),
15          .E_NE_N_tb),
16          .R_W(R_W_tb),
17          .Selec(Selec_tb),
18          .Num(Num_tb),
19          .Condicion(Condicion_tb),
20          .Operacion(Operacion_tb),
21          .Control_Salida(Control_Salida_tb),
22          .RegXY(RegXY_tb)
23      );
24
25      initial
26      begin
27          i_Instricciones_tb <= 9'b000000000; //No operacion
28          #10;
29          i_Instricciones_tb <= 9'b001010100; // Cargar en R2 el #4
30          #10;
31          i_Instricciones_tb <= 9'b010011101; // Cargar datos de [R5] en R3
32          #10;
33          i_Instricciones_tb <= 9'b011110111; // Almacenar en [R6] el #7
34          #10;
35          i_Instricciones_tb <= 9'b100100101; // Almacenar R5 en [R4]
36          #10;
37          i_Instricciones_tb <= 9'b101111001; // Mover R1 a R7
38          #10;
39          i_Instricciones_tb <= 9'b110011001; // Restar R0 menos R3
40          #10;
41          i_Instricciones_tb <= 9'b111010000; // Saltar Pc a R2 si condicion 0 es verdadera
42          #10;
43          i_Instricciones_tb <= 9'b000000000; //No operacion
44      end
45  endmodule
```

Testbench 2. Unidad de Control.

Banco_de_Registros_TB

```
1   module Banco_de_Registros_TB(
2       );
3       reg Clk_tb;
4       reg Rst_tb;
5       reg [7:0] Dat_tb;
6       reg [5:0] RegXY_tb;
7       reg [2:0] R_W_tb;
8       reg E_N_tb;
9       wire [7:0] RX_tb;
10      wire [7:0] RY_tb;
11
12      Banco_de_Registros DUT (
13          .Clk(Clk_tb),
14          .Rst(Rst_tb),
15          .Dat(Dat_tb),
16          .RegXY(RegXY_tb),
17          .R_W(R_W_tb),
18          .E_N(E_N_tb),
19          .RX(RX_tb),
20          .RY(RY_tb)
21      );
22
23      initial
24      begin
25          Clk_tb <= 0;
26          Rst_tb <= 1;
27          #10 Rst_tb <= 0;
28          Dat_tb <= 8'b00000101; // Dato a guardar = 5
29          RegXY_tb <= 6'b111110; // Salida registros= Mover = R6, Operanto = R7
30          R_W_tb <= 3'b001;      // Escribir en R1
31          E_N_tb <= 1;           // Habilitar la escritura
32          #10;
33          Dat_tb <= 8'b00000110; // Dato a guardar = 6
34          RegXY_tb <= 6'b001110; // Salida registros= Mover = R6, Operanto = R1
35          R_W_tb <= 3'b010;      // Escribir en R2
36          E_N_tb <= 1;           // Habilitar la escritura
37          #10;
38          Dat_tb <= 8'b00000111; // Dato a guardar = 7
39          RegXY_tb <= 6'b010010; // Salida registros= Mover = R2, Operanto = R2
40          R_W_tb <= 3'b100;      // Escribir en R4
```

```
41          E_N_tb <= 1;          // Habilitar la escritura
42          #10;
43          Dat_tb <= 8'b00000011; // Dato a guardar = 3
44          RegXY_tb <= 6'b100101; // Leer R4 Y R5
45          R_W_tb <= 3'b110;     // Escribir en R6
46          E_N_tb <= 0;          // Deshabilitar la escritura
47          #10;
48          Dat_tb <= 8'b00000111; // Dato a guardar = 7
49          RegXY_tb <= 6'b001010; // Leer R1 y R2
50          R_W_tb <= 3'b101;     // Escribir en R5
51          E_N_tb <= 0;          // Deshabilitar la escritura
52      end
53      always@(Clk_tb) begin
54          #5 Clk_tb <= ~Clk_tb;
55      end
56
57  endmodule
```

Testbench 3. Banco de Registros.

Unidad_de_Saltos_TB

```
1   module Unidad_de_Saltos_TB(
2       );
3       reg Clk_tb;
4       reg Rst_tb;
5       reg [7:0] Direccion_de_Salto_tb;
6       reg [2:0] Banderas_tb;
7       reg [3:0] Condicion_tb;
8       wire  [7:0] o_Bus_Direcciones_Instrucciones_tb;
9
10      Unidad_de_Saltos DUT (
11          .Clk(Clk_tb),
12          .Rst(Rst_tb),
13          .Direccion_de_Salto(Direccion_de_Salto_tb),
14          .Banderas(Banderas_tb),
15          .Condicion(Condicion_tb),
16          .o_Bus_Direcciones_Instrucciones(o_Bus_Direcciones_Instrucciones_tb)
17      );
18      initial
19      begin
20          Clk_tb<=0;
21          Rst_tb<=1;
22          Direccion_de_Salto_tb <= 8'b00000111;
23          Banderas_tb <= 3'b110; //Bandera de N y C activas
24          Condicion_tb <= 4'b0000; // Condicion inhabilitada
25          #10;
26          Rst_tb<=0;
27          #10;
28          Condicion_tb <= 4'b1000; //Evaluar condicion 0
29          #10;
30          Direccion_de_Salto_tb <= 8'b00000100;
31          Condicion_tb <= 4'b1001; //Evaluar condicion 1
32          #10;
33          Direccion_de_Salto_tb <= 8'b00001000;
34          Banderas_tb <= 3'b001; //Bandera Z activa
35          Condicion_tb <= 4'b1010; //Evaluar condicion 2
36          #10;
37          Direccion_de_Salto_tb <= 8'b00000010;
38          Condicion_tb <= 4'b1011; //Evaluar condicion 3;
39          #10;
40          Direccion_de_Salto_tb <= 8'b0001111;
```

```
41          Banderas_tb <= 3'b010; //Bandera C activa
42          Condicion_tb <= 4'b1100; //Evaluar condicion 4;
43          #10;
44          Direccion_de_Salto_tb <= 8'b00110010;
45          Condicion_tb <= 4'b1101; //Evaluar condicion 5;
46          #10;
47          Condicion_tb <= 4'b1110; //Evaluar condicion 6;
48          #10;
49          Condicion_tb <= 4'b1111; //Evaluar condicion 7;
50          #10;
51          Condicion_tb <= 4'b0000; // Condicion inhabilitada
52      end
53      always@(Clk_tb) begin
54          #5 Clk_tb<=~Clk_tb;
55      end
56  endmodule
```

Testbench 4. Unidad de Saltos.

Control_de_Bus_TB

```
1   module Control_de_Bus_TB(
2   );
3       reg Clk_tb;
4       reg Rst_tb;
5       reg [1:0] Control_Salida_tb;
6       reg [7:0] Num8b_tb;
7       reg [7:0] RX_tb;
8       reg [7:0] RY_tb;
9       wire [7:0] o_Bus_Datos_S_tb;
10      wire [7:0] o_Bus_Direccion_Datos_tb;
11      wire o_Lectura_Escritura_tb;
12              Control_de_Bus DUT(
13                      .Clk(Clk_tb),
14                      .Rst(Rst_tb),
15                      .Control_Salida(Control_Salida_tb),
16                      .Num8b(Num8b_tb),
17                      .RX(RX_tb),
18                      .RY(RY_tb),
19                      .o_Bus_Datos_S(o_Bus_Datos_S_tb),
20                      .o_Bus_Direccion_Datos(o_Bus_Direccion_Datos_tb),
21                      .o_Lectura_Escritura(o_Lectura_Escritura_tb)
22              );
23              initial
24              begin
25                      Clk_tb <= 0;
26                      Rst_tb <= 1;
27                      #10 Rst_tb <= 0;
28                      Num8b_tb <= 8'b00000011; //Num = 3
29                      RX_tb <= 8'b00001111; //RX = 15
30                      RY_tb <= 8'b00000001; //RY = 8
31                      Control_Salida_tb <= 2'b00; //0 en las salidas
32                      #10;
33                      Control_Salida_tb <= 2'b01; //RY como direccion
34                      #10;
35                      Control_Salida_tb <= 2'b10; //Escribir Num8b en direccion RX
36                      #10;
37                      Control_Salida_tb <= 2'b11; //Escribir RY en direccion RX
38                      #10;
39              end
40              always@(Clk_tb) begin
41                      #5 Clk_tb <= ~Clk_tb;
42              end
43  endmodule
```

Testbench 5. Control de Bus.

ALU_TB

```
1   module ALU_TB(
2   );
3           reg [7:0] R0_tb;
4           reg [7:0] RX_tb;
5          reg [3:0] Operacion_tb;
6          wire [7:0] Resultado_tb;
7           wire [2:0] Banderas_tb;
8          ALU DUT(
9                   .RX(RX_tb),
10                  .R0(R0_tb),
11                  .Operacion(Operacion_tb),
12                  .Resultado(Resultado_tb),
13                  .Banderas(Banderas_tb)
14         );
15         initial
16         begin
17            RX_tb <= 8'b00000011; // 3
18            R0_tb <= 8'b00000101; // 5
19                  Operacion_tb <= 4'b0000; //NO OPERACION
20                  #50; //50ns
21                  Operacion_tb <= 4'b1000; //SUMA R0 + RX
22                  #50; //50ns
23                  Operacion_tb <= 4'b1001; //RESTA R0 - RX
24                  #50; //50ns
25                  Operacion_tb <= 4'b1010; //DESPLAZAMIENTO IZQUIERDA R0 << RX
26                  #50; //50ns
27                  Operacion_tb <= 4'b1011; //DESPLAZAMIENTO DERECHA R0 >> RX
28                  #50; //50ns
29                 Operacion_tb <= 4'b1100; //NOT ~RX
30                 #50; //50ns
31                Operacion_tb <= 4'b1101; //AND RO&RX
32                 #50; //50ns
33                Operacion_tb <= 4'b1110; //OR R0|RX
34                #50; //50ns
35               Operacion_tb <= 4'b1111; //XOR R0^RX
36         end
37  endmodule
```

Testbench 6. ALU.

Mux1_TB

```
1   module Mux1_TB(
2   );
3           reg [7:0] Datos_0;
4           reg [7:0] Datos_1;
5           reg [7:0] Datos_2;
6           reg [7:0] Datos_3;
7           reg [7:0] Datos_4;
8           reg [2:0] Sel;
9           wire [7:0] Salida;
10          Mux1 DUT(
11                  .Resultado(Datos_3),
12                  .Num8b(Datos_2),
13                  .i_Bus_Datos_E(Datos_1),
14                  .Guardar_PC(Datos_0),
15                  .Mover(Datos_4),
16                  .Selec(Sel),
17                  .Dat(Salida)
18          );
19          initial
20          begin
21                  Datos_0<=8'b00010000; //Guardar_PC = 16
22                  Datos_1<= 8'b00100000; //i_Bus_Datos_E = 32
23                  Datos_2<= 8'b00000011; //Num8b = 3
24                  Datos_3<= 8'b11000000; //Resultado = 192
25                  Datos_4<= 8'b11110000; //Mover = 240
26                  Sel <= 3'b000; //Direcciona Guardar_PC a la salida
27                  #50;
28                  Sel <= 3'b001; //Direcciona i_Bus_Datos_E a la salida
29                  #50;
30                  Sel <= 3'b010; //Direcciona Num8b a la salida
31                  #50;
32                  Sel <= 3'b011; //Direcciona Resultado a la salida
33                  #50;
34                  Sel <= 3'b100; //Direcciona Mover a la salida
35                  #50;
36                  Sel <= 3'b101; //Direcciona un 0 a la salida
37                  #50;
38                 Sel <= 3'b110; //Direcciona un 0 a la salida
39                 #50;
40                 Sel <= 3'b001; //Direcciona i_Bus_Datos_E a la salida
41
42          end
43  endmodule
```

Testbench 7. Mux1.

ConNum_TB

```
1   module ConNum_TB(
2   );
3           reg [2:0] VNum;
4           wire [7:0] VNum8b;
5           ConNum DUT(
6                   .Num(VNum),
7                   .Num8b(VNum8b)
8           );
9           initial
10          begin
11                  VNum <= 3'b000;
12                  #50;
13                  VNum <= 3'b001;
14                  #50;
15                  VNum <= 3'b010;
16                  #50;
17                  VNum <= 3'b011;
18                  #50;
19                  VNum <= 3'b100;
20                  #50;
21                  VNum <= 3'b101;
22                  #50;
23                  VNum <= 3'b110;
24                  #50;
25                  VNum <= 3'b111;
26          end
27  endmodule
```

Testbench 8. ConNum.