



UNIVERSIDAD AUTÓNOMA DE ZACATECAS
“Francisco García Salinas”



UNIDAD ACADÉMICA DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN ROBÓTICA Y MECATRÓNICA

SISTEMAS DIGITALES III

Docente: Dr. Remberto Sandoval Aréchiga

Equipo 3: ROCKET

De Ávila Reveles Jorge Manuel

López Macías Geovanni

Salazar Rodríguez César Rodolfo

Grupo: 7º “B”

Reporte de MicroUAZ

Tabla de contenido

RESUMEN:	3
INTRODUCCIÓN	3
REQUERIMIENTOS.....	3
ARQUITECTURA.....	4
Diagrama de caja negra del Microcontrolador	4
Diagrama de caja Blanca del Microcontrolador.....	5
Diagrama de caja negra de MicroUAZ	5
Diagrama de caja blanca de MicroUAZ.....	7
Diagramas de caja negra de Módulo de Control	8
Diagrama de caja blanca de Módulo de Control.....	9
Diagramas de caja negra de Mux.....	10
Diagramas de caja negra de Registros	12
Diagramas de caja negra de ALU	14
Diagramas de caja negra de Administrador_de_Salidas	17
CONCLUSIONES.....	24
REFERENCIAS.....	24
Apéndice A: Códigos	25

RESUMEN:

Se elaboró un microcontrolador con una arquitectura tipo Harvard, el propósito del trabajo es implementarlo para hacer un algoritmo que realice una multiplicación y una división.

INTRODUCCIÓN

Todo sistema de cómputo tiene tres elementos básicos:

- Procesador: donde se procesan los datos.
- Memoria: lugar de almacenamiento de datos
- Dispositivos I/D : Con los cuales el procesador interactúa con el mundo real.

Los microprocesadores son el “cerebro” del computador: su centro lógico de operaciones aritméticas y lógicas, adonde van a ejecutarse todos los programas del sistema, tanto los propios del Sistema Operativo, como las aplicaciones ejecutadas por el usuario. Allí también se dan las lógicas binarias del sistema y los accesos a memoria. Es decir: el procesador es el motor informativo del computador.

Un microprocesador opera en base a una serie de instrucciones elementales que son pre programadas y almacenadas bajo la forma de código binario. Estas instrucciones van a organizarse a la memoria principal.

Se llama microprocesador o simplemente procesador al circuito integrado central de un sistema informática, en donde se llevan a cabo las operaciones lógicas y aritméticas (cálculos) para permitir la ejecución de los programas, desde el sistema operativo, hasta el software de aplicación.

REQUERIMIENTOS

- Conocer el panorama general de los microprocesadores y microcontroladores digitales, especialmente de los dispositivos de 8 bits. Identificar la estructura de un circuito de proceso lógico, así como de sus bloques internos y externos, las señales que requiere para funcionar, etc.
- Un bloque independiente de memoria para instrucciones y otro para datos, uno con un bus de dirección de 9 bits, la otra con un bus de dirección de 8 bits con una señal de escritura o lectura.
- Una señal de reseteo.
- Una señal de reloj.

ARQUITECTURA

Diagrama de caja negra del Microcontrolador

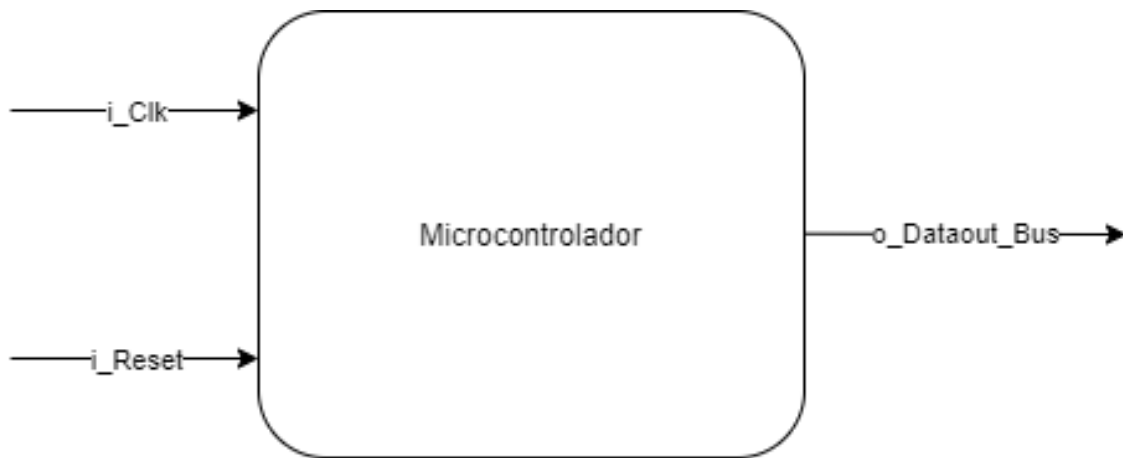


Figura 1. Caja negra de Microcotrolador, presentado sus entradas y salidas

Tabla de descripción

Nombre	Direccion	Num. de bits	Descripcion
i_Clk	Entrada	1	Señal de referencia de tiempo de 100Mhz
i_Reset	Entrada	1	Señal que restaura el micro a su estado inicial
o_Dataout_Bus	Salida	8	Señal generada que contiene los datos que queremos mostrar

Tabla 1. Tabla que muestra las señales de entrada y salida de la caja negra del microcontrolador

Diagrama de caja Blanca del Microcontrolador

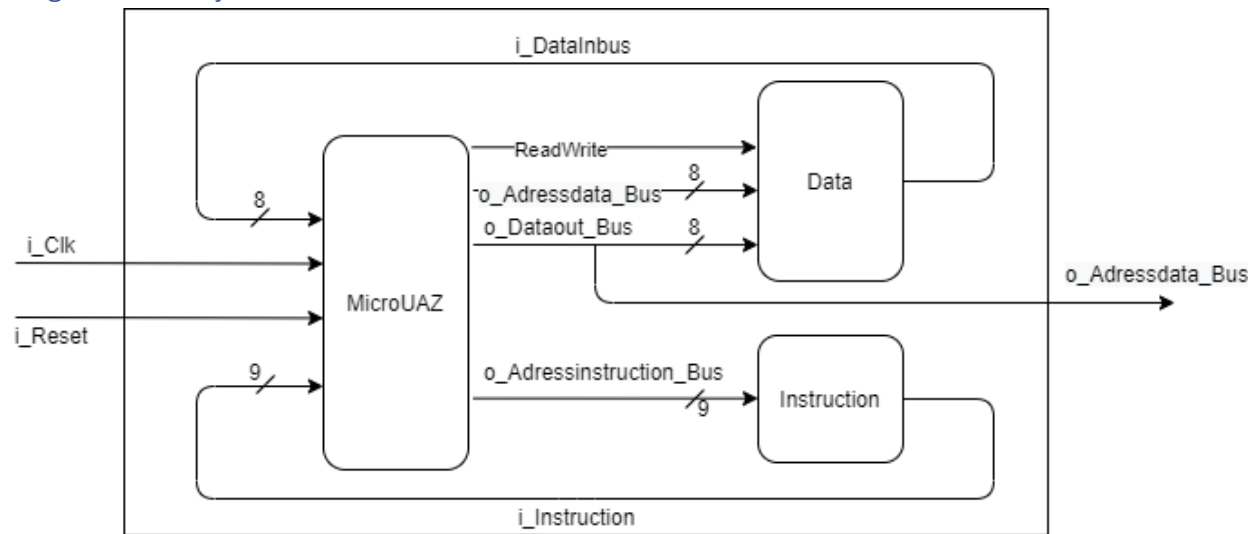


Figura 2. Caja blanca de Microcoontrolador, presentando los módulos internos

Tabla de descripción

Modulo	Descripción
MicroUAZ	Es un circuito lógico que procesa operaciones lógicas y aritméticas.
Data	Es la memoria que almacena datos durante algún periodo de tiempo.
Instruction	Es la memoria donde se almacenan las instrucciones del programa que debe ejecutar el microcontrolador.

Tabla 2. Tabla que muestra la descripción de los módulos contenidos en caja blanca del microcontrolador

Diagrama de caja negra de MicroUAZ

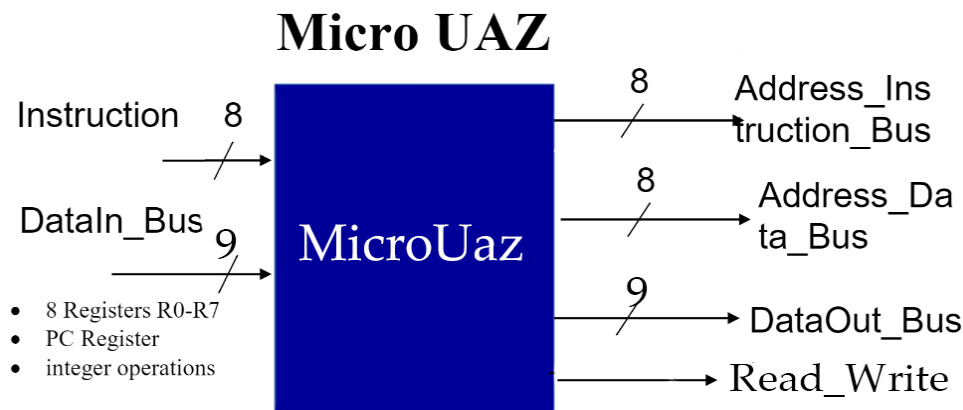


Figura 3. Caja negra de MicroUAZ, presentado sus entradas y salidas

Tabla de descripción

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: $R0=\sim RX$ 5: $R0=R0\&RX$ 6: $R0= R0 RX$ 7: $R0=R0\wedge RX$
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 3. Tabla que muestra las señales de entrada y salida de la caja negra de MicroUAZ

Diagrama de caja blanca de MicroUAZ

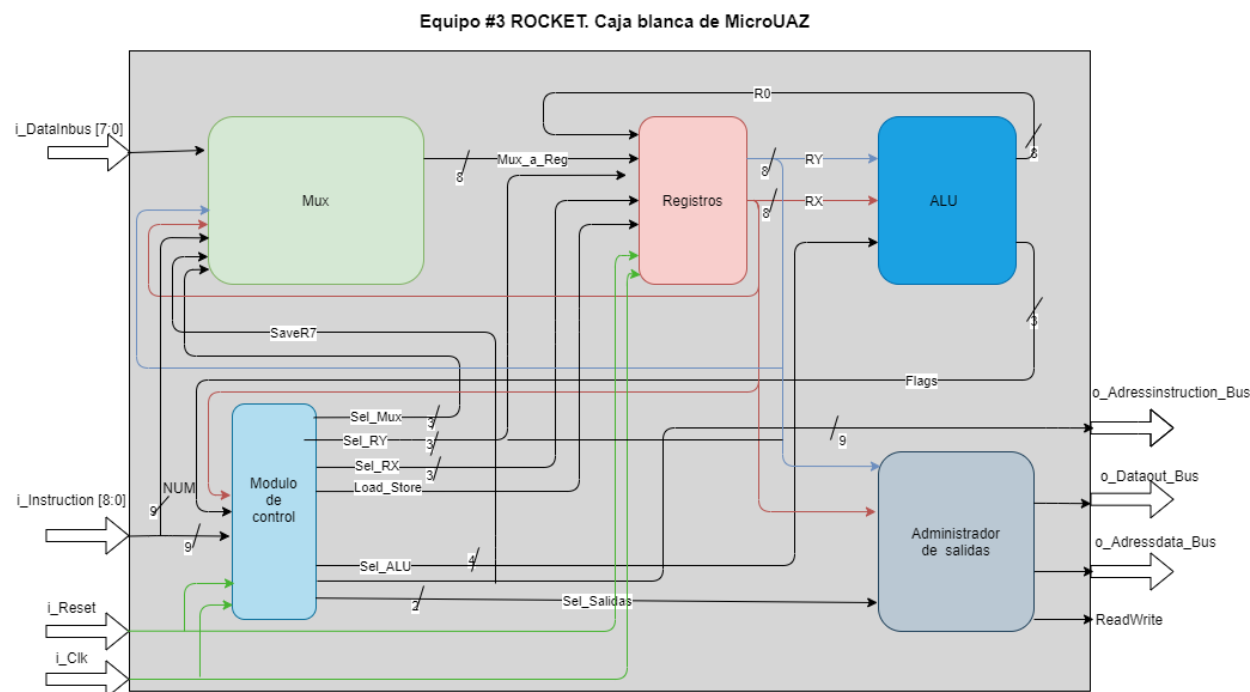


Figura 4. Caja negra de MicroUAZ, presentando sus módulos internos

Tabla de descripción

Modulo	Descripcion
Modulo de control	Se encarga de decodificar la entrada instruccion y de dividir las tarea a los diferentes modulos
Mux	Módulo que sirve como selector de entradas genera un valor que se asignara a registros
ALU	Unidad aritemetica logica del procesador
Registros	Módulo encargado de guardar y asignar registros
Administrador de Salidas	Módulo que administra las salidas

Tabla 4. Tabla que muestra la descripción de los módulos contenidos en caja blanca de MicroUAZ

Diagramas de caja negra de Módulo de Control

Caja negra de Módulo de control

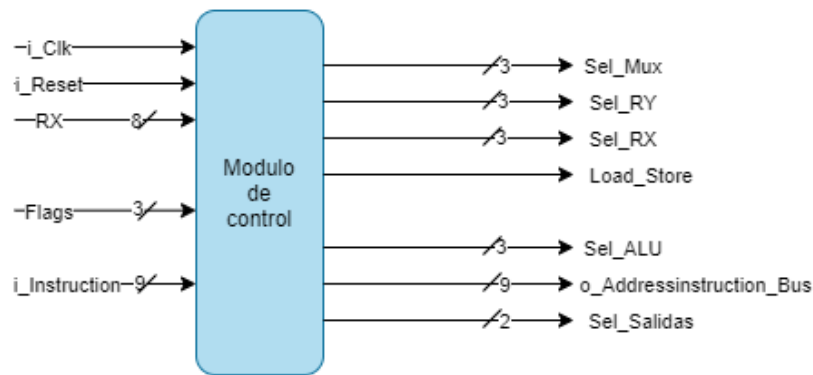


Figura 5. Caja negra de módulo de control, presentado sus entradas y salidas

Tabla de descripción

Nombre	Dirección	Num. de bits	Descripción
i_Clk	Entrada	1	Señal de referencia de tiempo de 100Mhz
i_Reset	Entrada	1	Señal que restaura el módulo a su estado inicial
RX	Entrada	8	Señal que contiene el valor del registro X
Flags	Entrada	3	Señal de 3 bits, cada uno de estos representa una bandera y cada una de estas puede estar en alto (1) o en bajo (0). Las banderas son generadas por el módulo ALU y son 3, ordenadas desde el bit menos significativo la más significativo son: Z (resultado igual a 0) ,C (resultado con acarreo), N (resultado negativo)
i_Instruction	Entrada	9	Señal de entrada recibida de memoria la cual será codificada.
Sel_Mux	Salida	3	Es una señal que sirve como selector en el módulo Mux
Sel_RY	Salida	3	Señal encargada de indicar cual de los 8 registros se carga en la salida RY
Sel_RX	Salida	3	Señal encargada de indicar cual de los 8 registros se carga en la salida RX
Load_Store	Salida	1	Señal que condiciona si haremos un load o un store
Sel_ALU	Salida	3	Señal que sirve para condicionar que operación se hará.
o_Addressinstruction_Bus	Salida	8	Señal encargada de enviar la instrucción
Sel_Salidas	Salida	2	Es una señal que sirve como selector en el módulo Administrador de salidas
ReadWrite	Salida	1	Señal de 1 bit que indica si se hará una lectura o una escritura sobre el registro. Si Write=0 se trata de una lectura Si Write=1 se trata de una escritura

Tabla 5. Tabla que muestra las señales de entrada y salida de la caja negra de Módulo de Control

Modulo que recibe las señal de o_instruction desde nuestra memoria y la decodificara haciendo así que de manera coordinada cada una de las acciones de los demás submodulos sean efectuadas.

I_instruction Instruccion, dato, dato [8:6],[5:3],[2:0]	Sel_Mux	Sel_Ry	Sel_Rx	LoadStore	Sel_ALU	Sel_Salidas	ControlJump	o_AddresInstruction
001,RX,Num	100	000	RX	1	000	00	000	o_AddresInstruction+1
010,[RY],RX	000	Ry	Rx	1	000	01	000	o_AddresInstruction+1
011,Rx,Num	100	000	Rx	0	000	10	000	o_AddresInstruction+1
100,[Rx],Ry	010	Ry	Rx	0	000	11	000	o_AddresInstruction+1
101,Rx,Ry	010	Ry	Rx	1	000	00	000	o_AddresInstruction+1
110,Rx,OP	010	001	Rx	0	OP	00	000	o_AddresInstruction+1
111,Rx,COND	101	000	000	0	000	00	COND	COND=0[i_RX] o_AddresInstruction+1 COND=1(SaveR7) o_AddresInstruction+1 COND=2, if(Z=1) out=i_RX COND=3, if(Z=0) out=i_RX COND=4, if(C=1) out=i_RX COND=5, if(C=0) out=i_RX COND=6, if(N=1) out=i_RX COND=7, if(N=0) out=i_RX
NOPE	000	000	000	0	000	00	000	o_AddresInstruction+1

Tabla 6. Tabla de verdad de Módulo de Control

Diagrama de caja blanca de Módulo de Control

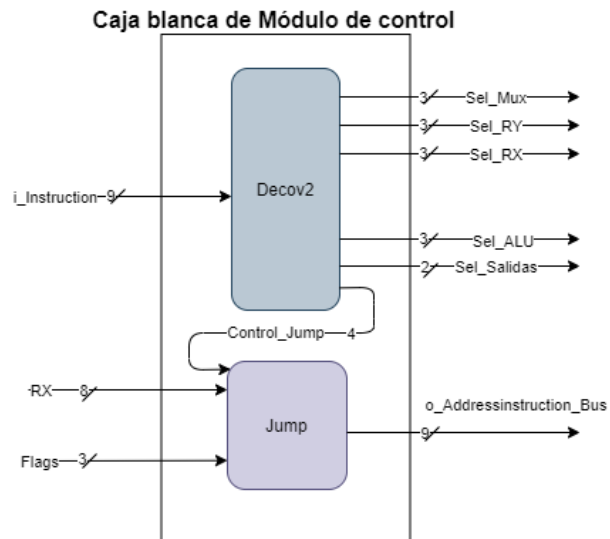


Figura 6. Caja blanca de módulo de control, presentando los módulos internos

Simulación de Decov2.v

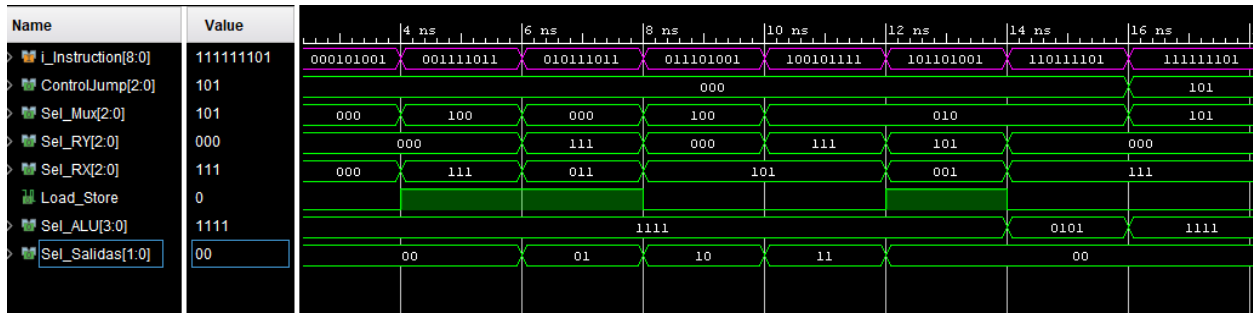


Figura 7. Simulación de Decov2

Como vemos en la figura 7 de color morado mostramos los cambios de las instrucciones que llegan de la señal i_Instruction el cual asigna diferentes valores a las señales (sel_Mux, Control_Jump, Sel_Ry, Sel_Rx, Load_Store, Sel_Alu, Sel_Salidas), el cambio se ve reflejado por la instrucción recibida.

Simulación de Jump

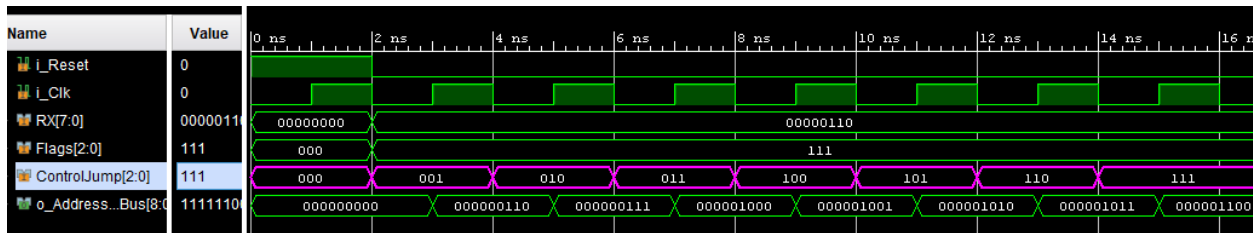


Figura 8. Simulación de Jump

Como se aprecia en la figura 8 tenemos una señal nombrada control jump de color rosa que se vera feectada por la señal de condición con el nombre flags.

Diagramas de caja negra de Mux

Caja negra de Mux



Figura 9. Caja negra del módulo Mux, presentado sus entradas y salidas

Tabla de descripción

Nombre	Direccion	Num. de bits	Descripcion
Num	Entrada	3	Señal recibida desde la memoria de instrucción
I_DataInBus	Entrada	8	Entrada de datos de la memoria
RX	Entrada	8	Señal que contiene el valor del registro X
RY	Entrada	8	Señal que contiene el valor del registro Y
Sel_Mux	Entrada	3	Es una señal que sirve como selector en el módulo Mux .
SaveR7	Entrada	9	Señal que contiene el valor de la entrada generada por la señal Sel_Mux
Mux_a_Reg	Salida	3	Señal que contiene el valor de la entrada generada por la señal Sel_Mux

Tabla 7. Descripción de señales del módulo Mux

Descripción funcional:

Modulo combinacional que sirve para seleccionar de varias entradas una única salida de datos. La señal Sel_Mux funge como selector y dependiendo de su valor de 3 bits podemos obtener las siguientes opciones:

Sel_Mux	Mux_a_Reg
000	I_DataBus
010	RY
011	RX
100	Num
101	SaveR7
Default	0

Tabla 8. Tabla de verdad del módulo Mux sensible a la señal Sel_Mux

Simulación de Mux

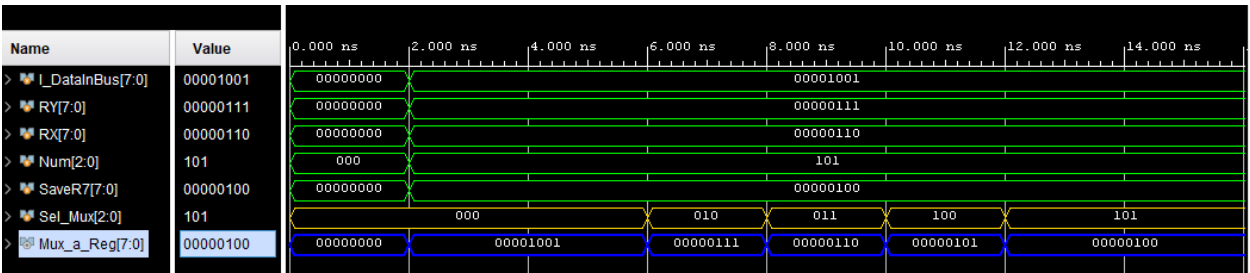


Figura 10. Simulación de Mux

Como se muestra en la Figura 10, tenemos una señal nombrada Sel_Mux de color amarillo, esta señal se encarga de seleccionar cual de nuestras señales de entrada (i_DataInBus, R0, RX, RY, Num y SaveR7) va a la salida nombrada Mux_a_Reg que está señalada de color azul.

Diagramas de caja negra de Registros



Figura 11. Caja negra del módulo Registros, presentado sus entradas y salidas

Tabla de descripción

Nombre	Direccion	Num. de bits	Descripcion
i_Clk	Entrada	1	Señal de referencia de tiempo de 100Mhz
i_Reset	Entrada	1	Señal que restaura el módulo a su estado inicial
R0	Entrada	8	Señal que indica el resultado de la operación realizada en la ALU
Mux_a_Registros	Entrada	8	Señal proveniente de Mux que contiene informacion que será almacenada en alguno de los registros
Sel_RY	Entrada	3	Señal encargada de indicar cual de los 8 registros se carga en la salida RY
Sel_RX	Entrada	3	Señal encargada de indicar cual de los 8 registros se carga en la salida RX
Load_Store	Entrada	1	Señal de 1 bit que indica si se hará una lectura o una escritura sobre el registro. Si Load_Store=0 se trata de una lectura Si Load_Store=1 se trata de una escritura
Sel_Registros	Entrada	3	Señal que indica en cual de los 8 registros (0-7) se va a almacenar el valor contenido en la señal Mux_a_Registros
RX	Salida	8	Señal que contiene el valor del registro X
RY	Salida	8	Señal que contiene el valor del registro Y

Tabla 9. Descripción de señales del modulo Registros

Descripción funcional

Se trata de un banco de registros encargado de almacenar registros para posteriormente enviarlos a la ALU, al Administrador de salidas o al Modulo de Control, dependiendo de la operación a utilizar. Tiene capacidad para almacenar 8 registros

Reset	Load_Store	RX	RY	Registros
0	0	Registro[Sel_RX]	Registro[Sel_RY]	Registros[0]=R0
0	1	Registro[Sel_RX]	Registro[Sel_RY]	Registros[Sel_RX]=Mux_a_Reg
1	0	0	0	0

Tabla 10. Tabla de verdad del módulo Registros sensible a las señales Load_Store

Simulación de Registros

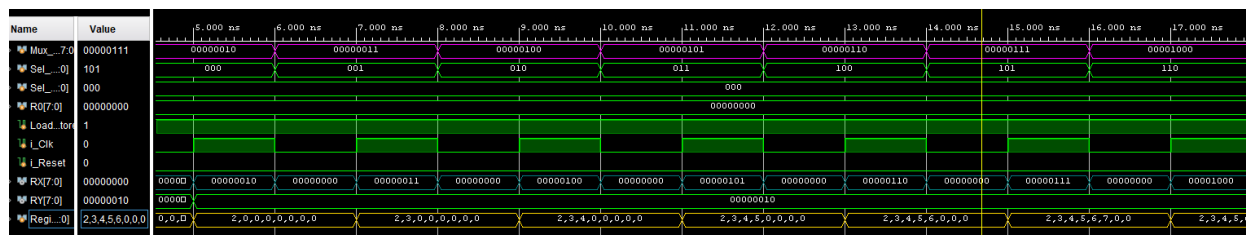


Figura 12. Simulación de Registros

Como se muestra en la Figura 12, mediante las señales Sel_Rx y Sel_Ry se selecciona que registro se asignara para cargar el valor que viene de la señal de entrada Mux_a_Reg, siempre y cuando load store este en alto, pero si la señal está en bajo asignara los valores a las señales de salida RX Y RY correspondientemente.

Diagramas de caja negra de ALU

Caja negra de ALU



Figura 11. Caja negra del módulo ALU, presentado sus entradas y salidas

Tabla de descripción

Nombre	Direccion	Num. de bits	Descripcion
RY	Entrada	8	Señal que contiene el registro X
RX	Entrada	8	Señal que contiene el registro Y
Sel_ALU	Entrada	4	Señal de 3 bits que indica que operación realizará la ALU.
R0	Salida	8	Señal que contiene el resultado de alguna de las operaciones realizada
Flags	Salida	3	Señal de 3 bits, cada uno de estos representa una bandera y cada una de estas puede estar en alto (1) o en bajo (0). Son 3, ordenadas desde el bit menos significativo la más significativo son: Z (resultado igual a 0), C (resultado con acarreo), N (resultado negativo)

Descripción funcional.

Tabla 11. Descripción de señales del modulo ALU

Módulo encargado de hacer las operaciones aritmético-lógicas del microprocesador. Esta módulo puede realizar las siguientes operaciones: Suma, resta, corrimiento a la izquierda, corrimiento a la derecha, NOT, AND, OR y XOR, dependiendo de la señal Sel_ALU como se indica a continuación:

Sel_ALU	R0
000	$R0 = RY + RX$
001	$R0 = RY - RX$
010	$R0 = RY \ll RX$
011	$R0 = RY \gg RX$
100	$R0 = \sim RX$
101	$R0 = RY \& RX$
110	$R0 = RY RX$
111	$R0 = RY \wedge RX$

Tabla 12. Tabla de verdad del módulo Mux sensible a la señal Sel_ALU

Simulación de ALU

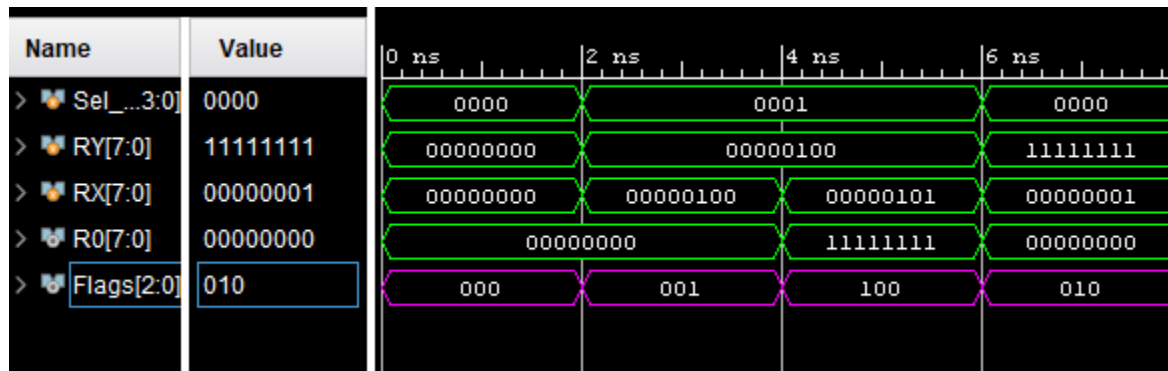


Figura 14. Simulación de ALU

Como se muestra en la Figura 14, mediante la señal Sel_ALU se seleccionará que operación se realizara con los valores de nuestras entradas RX y RY (la cual dependiendo del resultado de la operación este se asignara en R0) y se generara una señal nombrada Flags (de color rosa) la cual se encarga de indicarnos si hay o no hay un Cero (bit menos significativo), si hay o no hay Acarreo (segundo bit menos significativo), si hay o no hay valores Negativos (bit más significativo).

Caja negra de Administrador de salidas

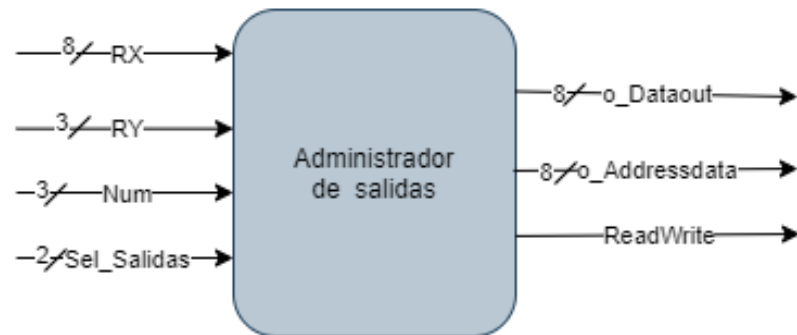


Figura 15. Caja negra del módulo ALU, presentado sus entradas y salidas

Tabla de descripción

Nombre	Direccion	Num. de bits	Descripcion
RX	Entrada	8	Señal de registro memoria generada para su posible uso
RY	Entrada	3	Señal de registro de memoria generara para su posible uso
Sel_Salidas	Entrada	2	Es una señal que sirve como selector del módulo
Num	Entrada	3	Señal que contiene el dato desde la memoria
o_Dataout	Salida	8	Señal generada que muestra los datos que queremos mostrar
o_Addressdata	Salida	9	Señal generada que va a la memoria nuestra ram
ReadWrite	Salida	1	Señal que se encarda de indicar si se va a leer o escribir en la memoria

Tabla 13. Descripción de señales del módulo Administrador de salidas

Sel_Salidas	o_Dataout	o_Addressdata	ReadWrite
00	0	0	0
01	0	RY	0
10	Num	RX	1
11	RY	RX	1

Tabla 14. Tabla de verdad del módulo Administrador de salidas sensible a la señal Sel_Salidas

Simulación de Administrador de Salidas

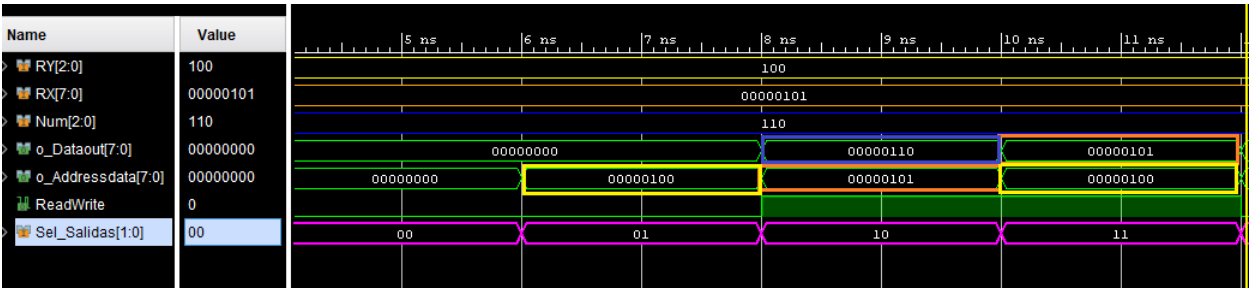


Figura 15. Simulación de Administrador_de_Salidas

En la Figura 15 tenemos la señal de control Sel_Salidas que se encarga de que señal o que señales saldrán por o_Dataout y o_Addressdata. Si Sel_salidas en 2'b01 el valor de RY se asignara a o_Addressdata si la señal es 2'b10 se asignara el valor de Num a o_Dataout y escribirá en memoria, 2'b11 RY sea nuestro Addressdata y RX será nuestro o_Dataout.

Diagrama de flujo de la Multiplicación

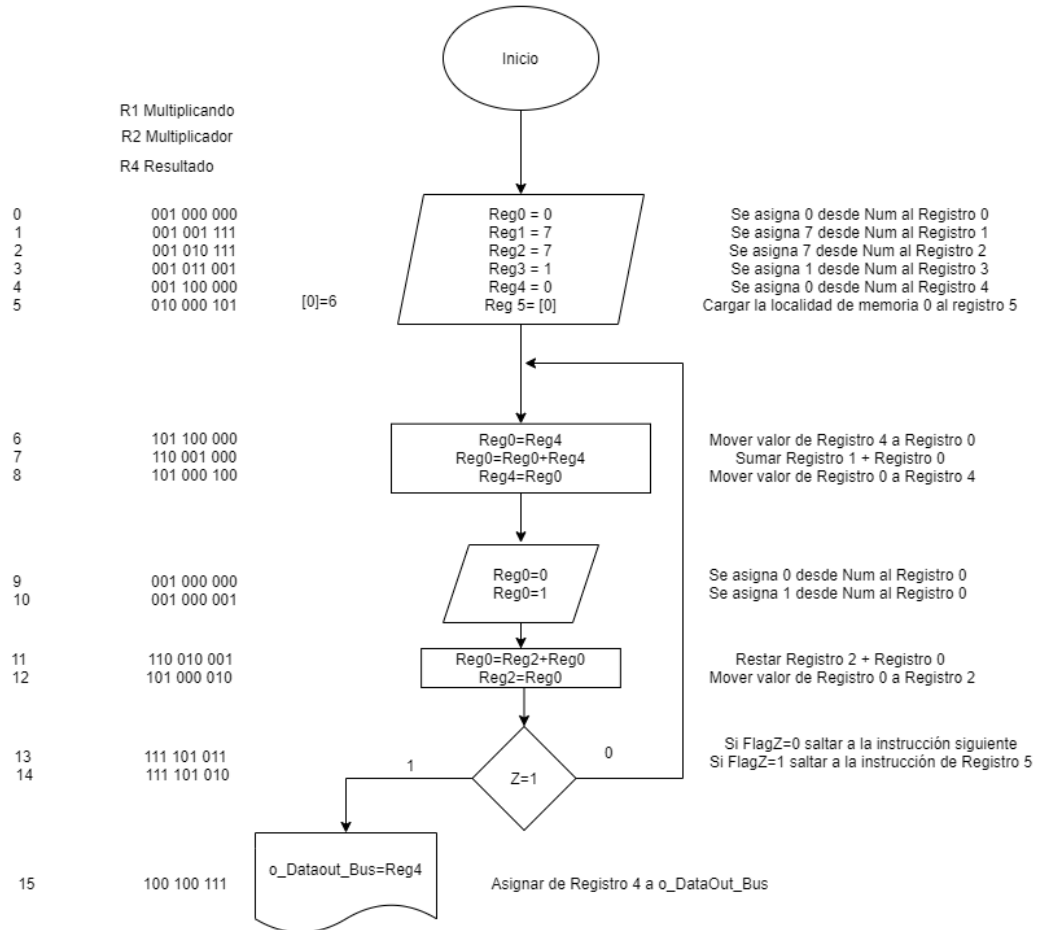


Figura 16. Diagrama de flujo de la multiplicación que muestra las instrucciones para realizar dicha operación.

Simulación de Multiplicación

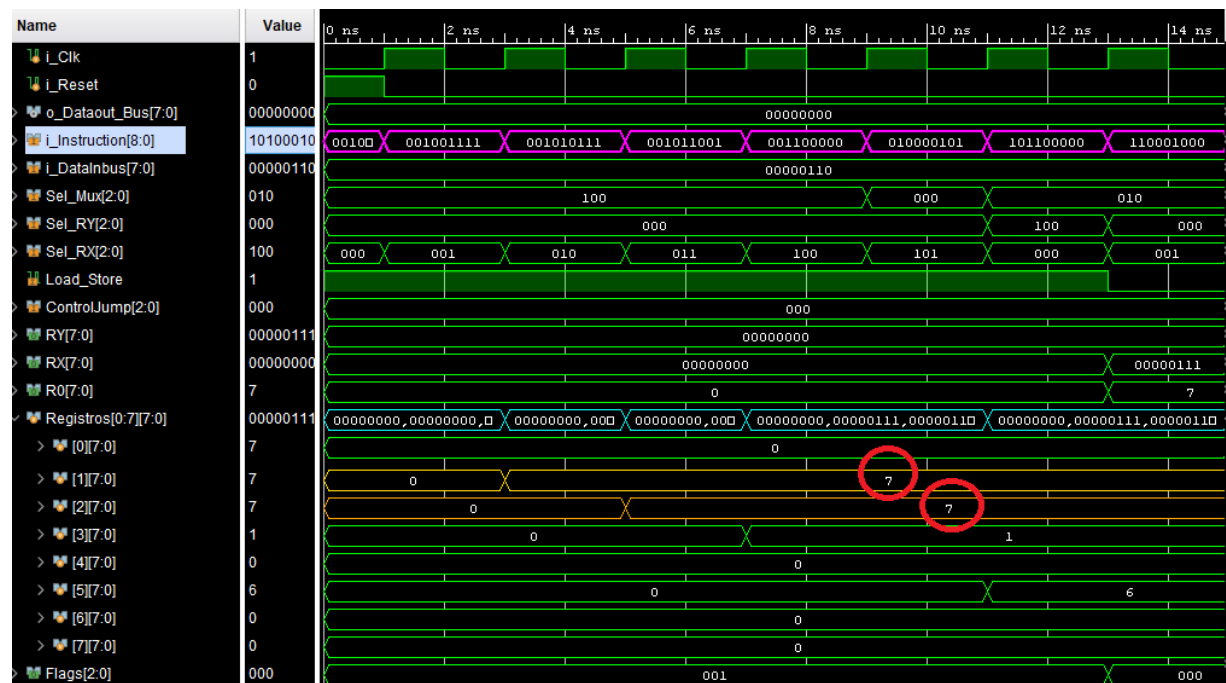


Figura 17. Simulación de Microcontrolador realizando la multiplicación, donde se muestran el multiplicador y el multiplicando.

En la Figura 17 se muestra como ejemplo una multiplicación en el multiplicando correspondiente al valor del registro 1 con un valor de 7 y el multiplicador se le asignó el mismo número correspondiente al registro 2 ambos señalizados con un círculo de color rojo.

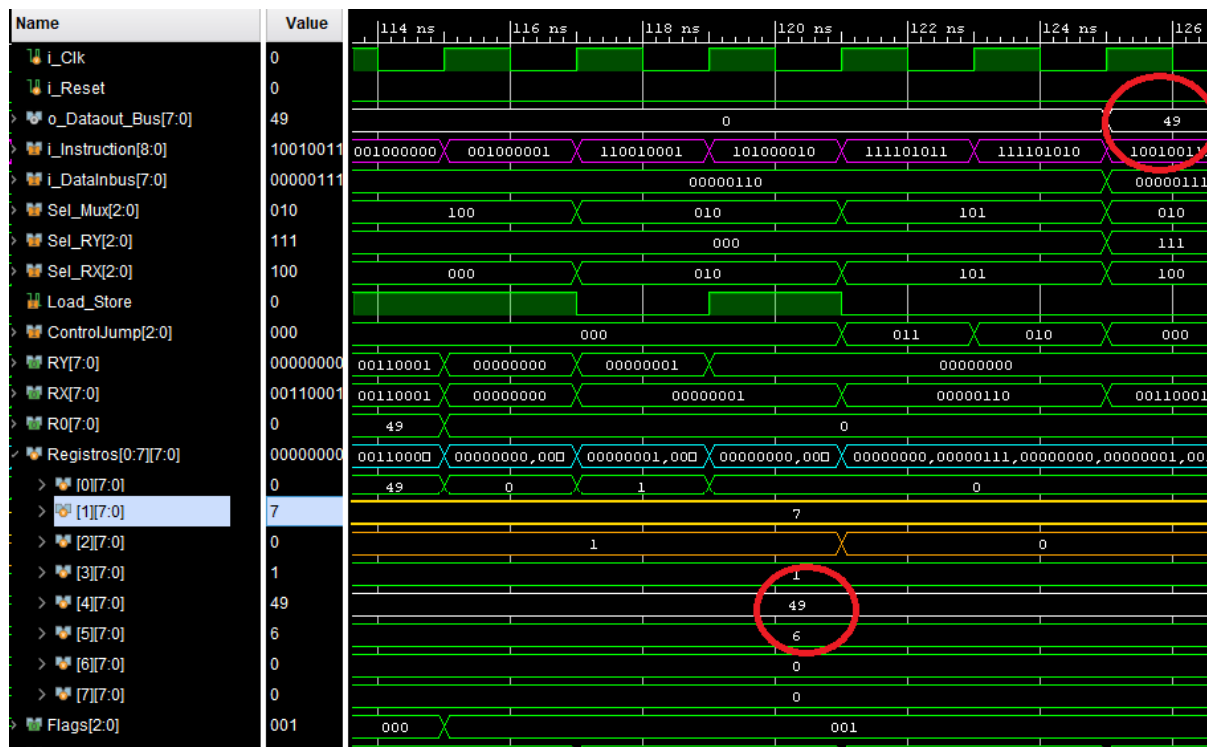


Figura 18. Simulación de Microcontrolador realizando la multiplicación, donde se muestra el producto de la multiplicación

En la Figura 18 se puede ver el resultado en el registro 4 y asignado a la señal o_Dataout_Bus

Diagrama de flujo de la División

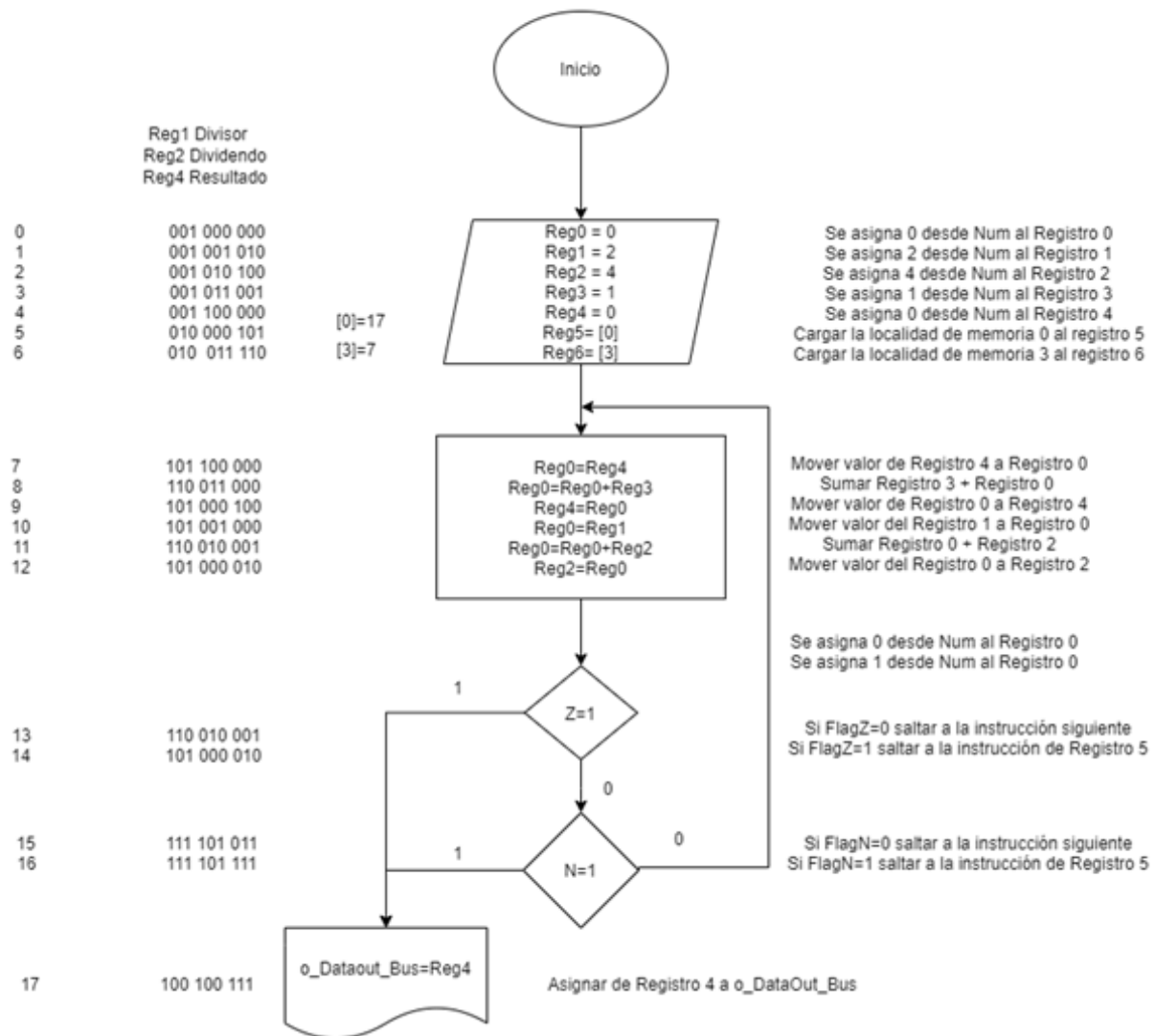


Figura 19. Diagrama de flujo de la división que muestra las instrucciones para realizar dicha operación.

Simulación de División

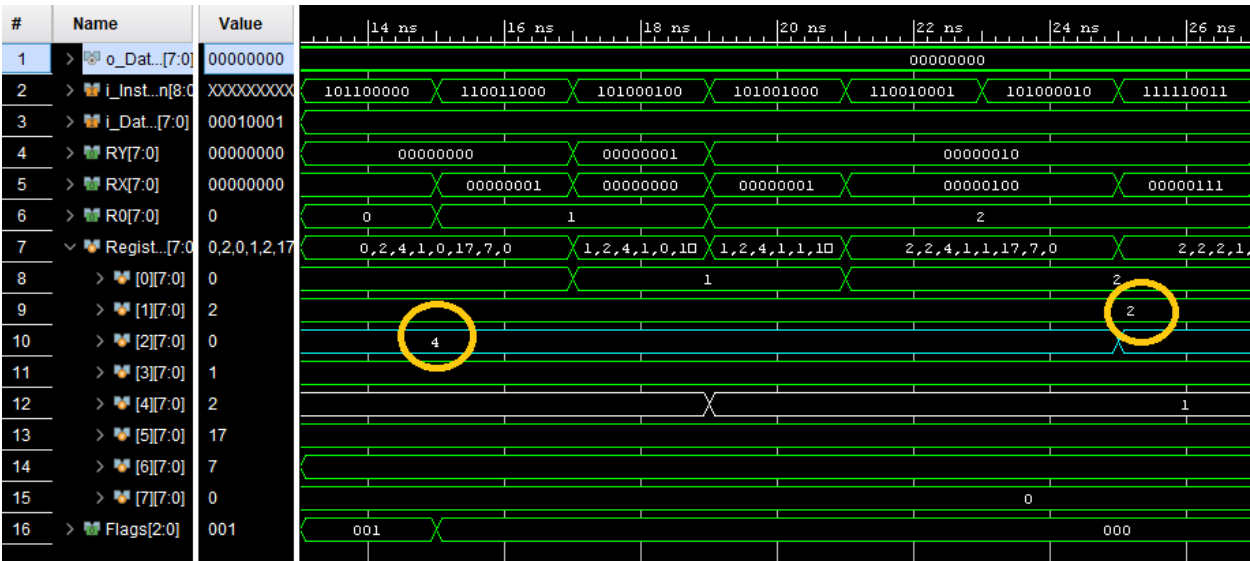


Figura 20. Simulación de Microcontrolador realizando la división, donde se muestran el divisor y el dividendo.

En la Figura 20 imagen se muestra como ejemplo una división en el divisor correspondiente al valor del registro 1 con un valor de 3 y el dividendo se le asignó el número 6 correspondiente al registro 2 ambos señalizados con un círculo de color amarillo.

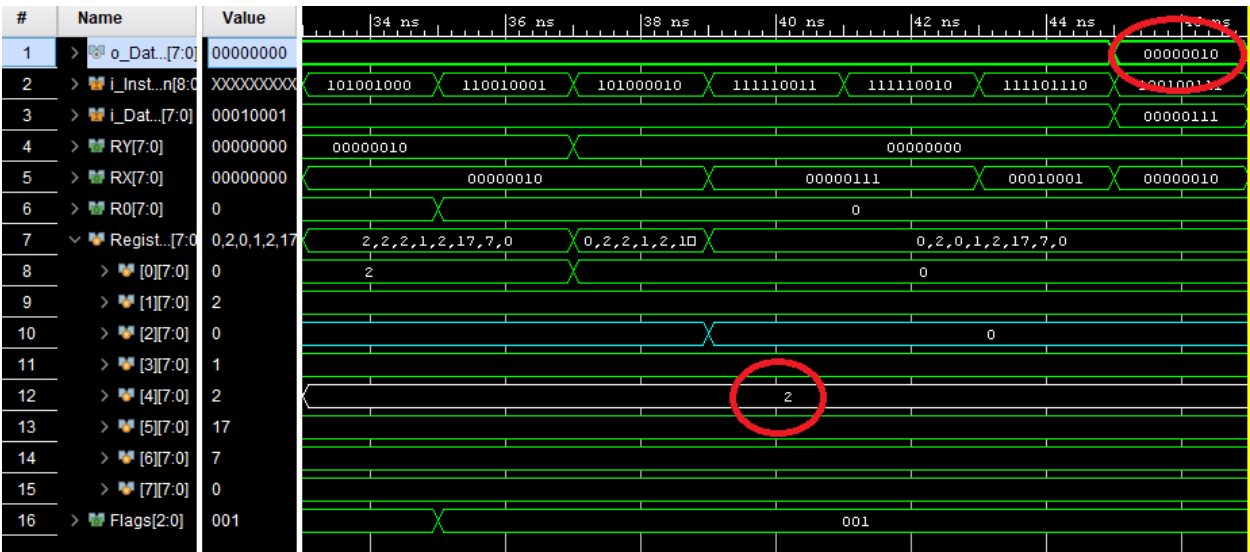


Figura 21. Simulación de Microcontrolador realizando la división, donde se muestra el cociente de la división

En la Figura 21 se puede ver el resultado en el registro 4 y asignado a la señal o_Dataout_Bus

CONCLUSIONES

A lo largo del desarrollo de este proyecto se presentaron tanto problemas del tipo técnico como social, al querer solucionar aquellos contratiempos algunas personas dejaron el proyecto, pero eso no bastó para poder finalizar lo que se nos pidió, al poner en prueba los conocimientos que obtuvimos de la materia en cuestión y de sus antecedentes, la metodología planteada fue fácil de interpretar por lo cual no hubo mayor problema el saber qué es lo que se nos pedía, y finalmente con las dudas aclaradas todo resultó en un proyecto funcional en la medida de lo que se pidió.

REFERENCIAS

[1] RAMÍREZ, Edward V. *“Introducción a los microprocesadores: equipo y sistemas.”* RWM Online, 1986

[2] L. Parra Reynada *“Elementos básicos de un microprocesador”* in *Microprocesadores*, Primera Edición, Estado de México, México, 2012 pp. 22. [Online] Available: <http://www.aliat.org.mx/BibliotecasDigitales/sistemas/Microprocesadores.pdf>

Apéndice A: Códigos

Código Decov2

```

23 module Decov2(
24     input [8:0] i_Instruction,
25
26     output reg [2:0] ControlJump,
27     output reg [2:0] Sel_Mux,
28     output reg [2:0] Sel_RY,
29     output reg [2:0] Sel_RX,
30     output reg Load_Store,
31     output reg [3:0] Sel_ALU,
32     output reg [1:0] Sel_Salidas
33 );
34
35     always@(i_Instruction)
36     begin
37         case (i_Instruction[8:6])
38             3'b001:
39                 begin
40                     Sel_Mux=3'b100;
41                     Sel_RY=3'b000;
42                     Sel_RX=i_Instruction[5:3];
43                     Load_Store=1'b1;
44                     Sel_ALU=4'b1111;
45                     Sel_Salidas=2'b00;
46                     ControlJump=3'b000;
47                 end
48             3'b010:
49                 begin
50                     Sel_Mux=3'b000;
51                     Sel_RY=i_Instruction[5:3];
52                     Sel_RX=i_Instruction[2:0];
53                     Load_Store=1'b1;
54                     Sel_ALU=4'b1111;
55                     Sel_Salidas=2'b01;
56                     ControlJump=3'b000;
57                 end
58             3'b011:
59                 begin
60                     Sel_Mux=3'b100;
61                     Sel_RY=3'b000;
62                     Sel_RX=i_Instruction[5:3];
63                     Load_Store=1'b0;
64                     Sel_ALU=4'b1111;
65                     Sel_Salidas=2'b10;
66                     ControlJump=3'b000;
67                 end
68             3'b100:
69                 begin
70                     Sel_Mux=3'b010;
71                     Sel_RY=i_Instruction[2:0];
72                     Sel_RX=i_Instruction[5:3];
73                     Load_Store=1'b0;
74                     Sel_ALU=4'b1111;
75                     Sel_Salidas=2'b11;
76                     ControlJump=3'b000;
77                 end
78             3'b101:
79                 begin
80                     Sel_Mux=3'b010;
81                     Sel_RY=i_Instruction[5:3];
82                     Sel_RX=i_Instruction[2:0];
83                     Load_Store=1'b1;
84                     Sel_ALU=4'b1111;
85                     Sel_Salidas=2'b00;
86                     ControlJump=3'b000;
87                 end
88             3'b110:
89                 begin
90                     Sel_Mux=3'b010;
91                     Sel_RY=3'b000;
92                     Sel_RX=i_Instruction[5:3];
93                     Load_Store=1'b0;
94                     Sel_ALU={1'b0,i_Instruction[2:0]};
95                     Sel_Salidas=2'b00;
96                     ControlJump=3'b000;
97                 end
98             3'b111:
99                 begin
100                     Sel_Mux=3'b101;
101                     Sel_RY=3'b000;
102                     Sel_RX=i_Instruction[5:3];
103                     Load_Store=1'b0;
104                     Sel_ALU=4'b1111;
105                     Sel_Salidas=2'b00;
106                     ControlJump=i_Instruction[2:0];
107                 end
108             default:
109                 begin
110                     Sel_Mux=0;
111                     Sel_RY=0;
112                     Sel_RX=0;
113                     Load_Store=0;
114                     Sel_ALU=4'b1111;
115                     Sel_Salidas=0;
116                     ControlJump=0;
117                 end
118         endcase
119     end

```

Código Jump

```
22 module Jump(  
23     input i_Reset,  
24     input i_Clk,  
25     input [7:0] RX,  
26     input [2:0] Flags,  
27     input [2:0] ControlJump,  
28     output reg [8:0] o_Addressinstruction_Bus  
29 );  
30  
31  
32     always@(posedge i_Clk, posedge i_Reset)  
33     begin  
34         if (i_Reset)  
35             begin  
36                 o_Addressinstruction_Bus=0;  
37             end  
38         else  
39             case(ControlJump)  
40                 3'b000: o_Addressinstruction_Bus=o_Addressinstruction_Bus+9'b000000001;  
41                 3'b001: o_Addressinstruction_Bus=RX;  
42                 3'b010:   
43                     begin  
44                         if (Flags[0])  
45                             o_Addressinstruction_Bus=o_Addressinstruction_Bus+9'b000000001;  
46                         else  
47                             o_Addressinstruction_Bus=RX;  
48                     end  
49                 3'b011:   
50                     begin  
51                         if (~Flags[0])  
52                             o_Addressinstruction_Bus=RX;  
53                         else  
54                             o_Addressinstruction_Bus=o_Addressinstruction_Bus+9'b000000001;  
55                     end  
56                 3'b100:   
57                     begin  
58                         if (Flags[1])  
59                             o_Addressinstruction_Bus=o_Addressinstruction_Bus+9'b000000001;  
60                         else  
61                             o_Addressinstruction_Bus=RX;  
62                     end  
63                 3'b101:   
64                     begin  
65                         if (~Flags[1])  
66                             o_Addressinstruction_Bus=RX;  
67                         else  
68                             o_Addressinstruction_Bus=o_Addressinstruction_Bus+9'b000000001;  
69                     end  
70                 3'b110:   
71                     begin  
72                         if (Flags[2])  
73                             o_Addressinstruction_Bus=o_Addressinstruction_Bus+9'b000000001;  
74                         else  
75                             o_Addressinstruction_Bus=RX;  
76                     end  
77                 3'b111:   
78                     begin  
79                         if (~Flags[2])  
80                             o_Addressinstruction_Bus=RX;  
81                         else  
82                             o_Addressinstruction_Bus=o_Addressinstruction_Bus+9'b000000001;  
83                     end  
84             endcase  
85         end  
86     end  
87  
88 endmodule
```

Código Modulo_de_Control

```
22 module Modulo_de_Control(  
23     input [7:0] RX,  
24     input [2:0] Flags ,  
25     input [8:0] i_Instruction,  
26     input i_Clk,  
27     input i_Reset,  
28     output [2:0] Sel_Mux,  
29     output [2:0] Sel_RY,  
30     output [2:0] Sel_RX,  
31     output Load_Store,  
32     output [2:0] Sel_Registros,  
33     output [3:0] Sel_ALU,  
34     output [8:0] o_Addressinstruction_Bus,  
35     output [1:0] Sel_Salidas  
36 );  
37  
38 wire [2:0] ControlJump;  
39  
40 Jump Brinco(  
41     .ControlJump(ControlJump),  
42     .RX(RX),  
43     .Flags(Flags),  
44     .i_Clk(i_Clk),  
45     .i_Reset(i_Reset),  
46     .o_Addressinstruction_Bus(o_Addressinstruction_Bus)  
47 );  
48  
49 Decov2 Decodificador(  
50     .i_Instruction(i_Instruction),  
51     .Sel_Mux(Sel_Mux),  
52     .Sel_RY(Sel_RY),  
53     .Sel_RX(Sel_RX),  
54     .Load_Store(Load_Store),  
55     .Sel_ALU(Sel_ALU),  
56     .Sel_Salidas(Sel_Salidas),  
57     .ControlJump(ControlJump)  
58 );  
59  
60 endmodule
```

Código Mux

```
22 module Mux(  
23     input [7:0] i_DataInBus,  
24     input [7:0] RY,  
25     input [7:0] RX,  
26     input [2:0] Num,  
27     input [8:0] SaveR7,  
28     input [2:0] Sel_Mux,  
29     output reg [7:0] Mux_a_Reg  
30 );  
31  
32 always@(*)  
33 begin  
34     case (Sel_Mux)  
35         3'b000: Mux_a_Reg = i_DataInBus;  
36         3'b010: Mux_a_Reg = RY;  
37         3'b011: Mux_a_Reg = RX;  
38         3'b100: Mux_a_Reg = Num;  
39         3'b101: Mux_a_Reg = SaveR7;  
40         default: Mux_a_Reg <= 0;  
41     endcase  
42 end  
43 endmodule
```

Código Registros

```
23 module Registros(  
24     input i_Reset,  
25     input i_Clk,  
26     input [7:0] R0,  
27     input [2:0] Sel_RX,  
28     input [2:0] Sel_RY,  
29     input Load_Store, // 0->leer, 1->escribir  
30     input [7:0] Mux_a_Reg,  
31     output [7:0] RX,  
32     output [7:0] RY  
33 );  
34  
35 reg [7:0] Registros [0:7];  
36 reg [7:0] ry;  
37 reg [7:0] rx;  
38 always @(posedge i_Clk, posedge i_Reset) begin  
39     if(i_Reset)  
40         begin  
41             Registros[0]<=0;  
42             Registros[1]<=0;  
43             Registros[2]<=0;  
44             Registros[3]<=0;  
45             Registros[4]<=0;  
46             Registros[5]<=0;  
47             Registros[6]<=0;  
48             Registros[7]<=0;  
49         end  
50     else  
51         if (Load_Store)  
52             Registros[Sel_RX[2:0]]<=Mux_a_Reg;  
53         else  
54             Registros[0]=R0;  
55         end  
56         assign RX=Registros[Sel_RX[2:0]];  
57         assign RY=Registros[Sel_RY[2:0]];  
58     endmodule
```

Código ALU

```

22 module ALU #(parameter N=8) (
23     input [7:0] RY,
24     input [7:0] RX,
25     input wire [3:0] Sel_ALU,
26     output [7:0] R0,
27     output [2:0] Flags
28 );
29
30 wire[N-1:0]salidasuma;
31 wire[N-1:0]salidaresta;
32 wire[N-1:0]salidaDizquierda;
33 wire[N-1:0]salidaDderecha;
34 wire[N-1:0]salidanot;
35 wire[N-1:0]salidaand;
36 wire[N-1:0]salidaor;
37 wire[N-1:0]salidaxor;
38 wire[N-1:0]salidanada;
39 wire[3:0] Suma_o_Resta;
40 wire c_out_suma;
41 wire c_out_resta;
42
43
44 SumadorNbits #(N(N)) Op0(
45     .a(RX),
46     .b(RY),
47     .c in(1'b0),
48     .sum(salidasuma),
49     .c_out(c_out_suma)
50 );
51
52 Op_Resta #(N(N)) Op1(
53     .A(RY),
54     .B(RX),
55     .R(salidaresta)
56 );
57
58 Desplazamiento_Izquierda#(N(N)) Op2(
59     .A(RX),
60     .Y(salidaDizquierda)
61 );
62
63 Desplazamiento_Derecha#(N(N)) Op3(
64     .A(RY),
65     .Y(salidaDderecha)
66 );
67
68
69 Op_Not#(N(N)) Op4(
70     .A(RX),
71     .Y(salidanot)
72 );
73
74 Op_And#(N(N)) Op5(
75     .A(RX),
76     .B(RY),
77     .Y(salidaand)
78 );
79
80 Op_Or#(N(N)) Op6(
81     .A(RX),
82     .B(RY),
83     .Y(salidaor)
84 );
85
86 Nada#(N(N)) Op7(
87     .A(RX),
88     .Y(salidanada)
89 );
90
91 Op_Xor#(N(N)) Op7(
92     .A(RX),
93     .B(RY),
94     .Y(salidaxor)
95 );
96
97 Mux_ALU #(N(N)) MuxALU(
98     .salidasuma(salidasuma),
99     .salidaresta(salidaresta),
100     .salidaDizquierda(salidaDizquierda),
101     .salidaDderecha(salidaDderecha),
102     .salidanot(salidanot),
103     .salidaand(salidaand),
104     .salidaor (salidaor),
105     .salidaxor(salidaxor),
106     .Operador(Sel_ALU),
107     .SalidaOp(R0),
108     .OperadorSalida(Suma_o_Resta),
109     .salidanada(salidanada)
110 );
111
112 Flag_Indicator #(N(N)) Banderas(
113     .A(RY),
114     .B(RX),
115     .SalidaOp(R0),
116     .c_out_suma(c_out_suma),
117     .Suma_o_Resta(Suma_o_Resta),
118     .Flags (Flags)
119 );
120
121 endmodule

```

Código Administrador_de_salidas

```
23 module Administrador_de_salidas(  
24     input [2:0] RY,  
25     input [7:0] RX,  
26     input [2:0] Num,  
27     input [1:0] Sel_Salidas,  
28     output reg [7:0] o_Dataout,  
29     output reg [7:0] o_Addressdata,  
30     output reg ReadWrite  
31 );  
32  
33 always@*  
34 begin  
35     case(Sel_Salidas)  
36     2'b00:  
37         begin  
38             o_Dataout <= 0;  
39             o_Addressdata <= 0;  
40             ReadWrite <= 0;  
41         end  
42     2'b01:  
43         begin  
44             o_Dataout <= 0;  
45             o_Addressdata <= {5'b00000,RY};  
46             ReadWrite <= 0;  
47         end  
48     2'b10:  
49         begin  
50             o_Dataout <= {5'b00000,Num};  
51             o_Addressdata <= RX;  
52             ReadWrite <= 1;  
53         end  
54     2'b11:  
55  
56         begin  
57             o_Dataout <= RX;  
58             o_Addressdata <= RY;  
59             ReadWrite <= 1;  
60         end  
61     default  
62         begin  
63             o_Dataout <= 0;  
64             o_Addressdata <= 0;  
65             ReadWrite <= 0;  
66         end  
67     endcase  
68 end  
69  
70 endmodule
```

Código MicroUAZ

```

23 module MicroUAZ(
24     input [7:0] i_DataInbus,
25     input [8:0] i_Instruction,
26     input i_Clk,
27     input i_Reset,
28     output [8:0] o_Addressinstruction_Bus,
29     output [7:0] o_Dataout_Bus,
30     output [7:0] o_Addressdata_Bus,
31     output ReadWrite
32 );
33
34 wire [2:0] Sel_Mux;
35 wire [2:0] Sel_RY;
36 wire [2:0] Sel_RX;
37 wire Load_Store;
38 wire [2:0] w_Sel_Registros;
39 wire [3:0] Sel_ALU;
40 wire [1:0] Sel_Salidas;
41 wire [7:0] Mux_a_Reg;
42 wire [7:0] R0;
43 wire [7:0] RY;
44 wire [7:0] RX;
45 wire [2:0] Flags;
46
47 Mux Multiplexor(
48     .i_DataInBus(i_DataInbus),
49     .RY(RY),
50     .RX(RX),
51     .Num(i_Instruction),
52     .SaveR7(o_Addressinstruction_Bus),
53     .Sel_Mux(Sel_Mux),
54     .Mux_a_Reg(Mux_a_Reg)
55 );
56
57 Modulo_de_Control Control(
58     .RX(RX),
59     .Flags(Flags),
60     .i_Instruction(i_Instruction),
61     .i_Clk(i_Clk),
62     .i_Reset(i_Reset),
63     .Sel_Mux(Sel_Mux),
64     .Sel_RY(Sel_RY),
65     .Sel_RX(Sel_RX),
66     .Load_Store(Load_Store),
67     .Sel_ALU(Sel_ALU),
68     .o_Addressinstruction_Bus(o_Addressinstruction_Bus),
69     .Sel_Salidas(Sel_Salidas)
70 );
71
72 Registros Banco_de_Registros(
73     .R0(R0),
74     .Mux_a_Reg(Mux_a_Reg),
75     .Sel_RY(Sel_RY),
76     .Sel_RX(Sel_RX),
77     .Load_Store(Load_Store),
78     .i_Clk(i_Clk),
79     .i_Reset(i_Reset),
80     .RY(RY),
81     .RX(RX)
82 );
83
84 ALU UnidadAL(
85     .RY(RX),
86     .RX(RY),
87     .Sel_ALU(Sel_ALU),
88     .R0(R0),
89     .Flags(Flags)
90 );
91
92 Administrador_de_salidas Salidas(
93     .RY(Sel_RY),
94     .RX(RX),
95     .Num(i_Instruction),
96     .Sel_Salidas(Sel_Salidas),
97     .o_Dataout(o_Dataout_Bus),
98     .o_Addressdata(o_Addressdata_Bus),
99     .ReadWrite(ReadWrite)
100 );
101
102 endmodule

```


Código Data

```
22 module Data(  
23     input [7:0] i_DataInbus,  
24     input [7:0] o_Dataout,  
25     input ReadWrite,  
26     output reg [7:0] o_Addressdata_Bus  
27 );  
28     reg [7:0] mem [0:255];  
29  
30     initial  
31     begin  
32         $readmemb("Data.mem",mem);  
33     end  
34     always@(i_DataInbus,o_Dataout,ReadWrite)  
35     begin  
36         if(ReadWrite)  
37         begin  
38             mem[i_DataInbus]<=o_Dataout;  
39             o_Addressdata_Bus<=mem[i_DataInbus];  
40         end  
41         else  
42             o_Addressdata_Bus<=mem[i_DataInbus];  
43     end  
44 endmodule
```

Código Instruction

```
23 module Instruction(  
24     input [8:0] i_Instruction,  
25     output reg [8:0] o_Addressinstruction_Bus  
26 );  
27     reg [8:0] mem [0:255];  
28  
29     initial  
30     $readmemb("Instruction.mem",mem,0,255);  
31     always@(i_Instruction)  
32     o_Addressinstruction_Bus <= mem[i_Instruction];  
33  
34 endmodule
```

Código Microcontrolador

```
23 module Microcontrolador(  
24     input  i_Clk,  
25     input  i_Reset,  
26     output [7:0] o_Dataout_Bus  
27 );  
28     wire [7:0] i_DataInbus;  
29     wire [8:0] i_Instruction;  
30     wire [8:0] o_Addressinstruction_Bus;  
31     wire [7:0] o_Addressdata_Bus;  
32     wire ReadWrite;  
33  
34     MicroUAZ Micro(  
35         .i_DataInbus(i_DataInbus),  
36         .i_Instruction(i_Instruction),  
37         .i_Clk(i_Clk),  
38         .i_Reset(i_Reset),  
39         .o_Addressinstruction_Bus(o_Addressinstruction_Bus),  
40         .o_Dataout_Bus(o_Dataout_Bus),  
41         .o_Addressdata_Bus(o_Addressdata_Bus),  
42         .ReadWrite(ReadWrite)  
43     );  
44  
45     Data Ram(  
46         .i_DataInbus(o_Addressdata_Bus),  
47         .o_Dataout(o_Dataout_Bus),  
48         .o_Addressdata_Bus(i_DataInbus),  
49         .ReadWrite(ReadWrite)  
50     );  
51  
52     Instruction Rom(  
53         .i_Instruction(o_Addressinstruction_Bus),  
54         .o_Addressinstruction_Bus(i_Instruction)  
55     );  
56  
57 endmodule
```

Códigos de los TestBench

Código Tb_Deco

```
22 module Tb_Deco;
23     reg [8:0] i_Instruction;
24
25     wire [2:0] ControlJump;
26     wire [2:0] Sel_Mux;
27     wire [2:0] Sel_RY;
28     wire [2:0] Sel_RX;
29
30     wire Load_Store;
31
32     wire [2:0] Sel_ALU;
33     wire [1:0] Sel_Salidas;
34
35     Decov2 uut(
36         .i_Instruction(i_Instruction),
37         .ControlJump(ControlJump),
38         .Sel_Mux(Sel_Mux),
39         .Sel_RY(Sel_RY),
40         .Sel_RX(Sel_RX),
41         .Load_Store(Load_Store),
42         .Sel_ALU(Sel_ALU),
43         .Sel_Salidas(Sel_Salidas)
44     );
45
46     initial
47     begin
48
49         i_Instruction=0;
50
51         #2 i_Instruction=9'b000101001;
52         #2 i_Instruction=9'b001111011;
53         #2 i_Instruction=9'b010111011;
54         #2 i_Instruction=9'b011101001;
55         #2 i_Instruction=9'b100101111;
56         #2 i_Instruction=9'b101101001;
57         #2 i_Instruction=9'b110111101;
58         #2 i_Instruction=9'b111111101;
59
60     end
61 endmodule
```

Código Tb_Jump

```

22 module Tb_Jump(
23
24 );
25 reg i_Clk;
26 reg i_Reset;
27 reg [7:0] RX;
28 reg [2:0] Flags;
29 reg [2:0] ControlJump;
30 wire [8:0] o_Addressinstruction_Bus;
31
32 Jump uut (
33     .i_Reset(i_Reset),
34     .i_Clk(i_Clk),
35     .RX(RX),
36     .Flags(Flags),
37     .ControlJump(ControlJump),
38     .o_Addressinstruction_Bus(o_Addressinstruction_Bus)
39 );
40
41 initial
42 begin
43
44     i_Reset=1;
45     i_Clk=0;
46     RX=0;
47     Flags=3'b000;
48     ControlJump=3'b000;
49
50     #2
51     i_Reset=0;
52     RX=6;
53     Flags=3'b111;
54     ControlJump=3'b001;
55
56
57     #2
58     RX=6;
59     Flags=3'b111;
60     ControlJump=3'b010;
61
62     #2
63     RX=6;
64     Flags=3'b111;
65     ControlJump=3'b011;
66
67     #2
68     RX=6;
69     Flags=3'b111;
70     ControlJump=3'b100;
71
72     #2
73     RX=6;
74     Flags=3'b111;
75     ControlJump=3'b101;
76
77     #2
78     RX=6;
79     Flags=3'b111;
80     ControlJump=3'b110;
81
82     #2
83     RX=6;
84     Flags=3'b111;
85     ControlJump=3'b111;
86
87 end
88 always
89     #1 i_Clk = !i_Clk;
90 endmodule

```

Código Tb_Mux

```
22 module Tb_Mux;
23     reg [7:0] I_DataInBus;
24     reg [7:0] RY;
25     reg [7:0] RX;
26     reg [2:0] Num;
27     reg [7:0] SaveR7;
28     reg [2:0] Sel_Mux;
29     wire [7:0] Mux_a_Reg;
30
31     Mux uut(
32         .i_DataInBus(I_DataInBus),
33         .RY(RY),
34         .RX(RX),
35         .Num(Num),
36         .SaveR7(SaveR7),
37         .Sel_Mux(Sel_Mux),
38         .Mux_a_Reg(Mux_a_Reg)
39     );
40     initial
41     begin
42         I_DataInBus=0;
43         RY=0;
44         RX=0;
45         Num=0;
46         SaveR7=0;
47         Sel_Mux=3'b000;
48         #2 I_DataInBus=8'b00001001; RY=8'b00000111;
49         RX=8'b00000110; Num=8'b00000101; SaveR7=8'b00000100;
50         #2 Sel_Mux=3'b000;
51         #2 Sel_Mux=3'b010;
52         #2 Sel_Mux=3'b011;
53         #2 Sel_Mux=3'b100;
54         #2 Sel_Mux=3'b101;
55     end
56 endmodule
```

Código Tb_Registros

```
22 module Tb_Registros;
23     reg [7:0] Mux_a_Reg;
24     reg [2:0] Sel_RX;
25     reg [2:0] Sel_RY;
26     reg [7:0] R0;
27     reg Load_Store;
28     reg i_Clk;
29     reg i_Reset;
30     wire [7:0] RX;
31     wire [7:0] RY;
32
33     Registros uut(
34         .Mux_a_Reg(Mux_a_Reg),
35         .Sel_RX(Sel_RX),
36         .Sel_RY(Sel_RY),
37         .R0(R0),
38         .Load_Store(Load_Store),
39         .i_Clk(i_Clk),
40         .i_Reset(i_Reset),
41         .RX(RX),
42         .RY(RY)
43     );
44
45     initial
46     begin
47         i_Reset=1;
48         i_Clk=0;
49         Sel_RX=0;
50         Sel_RY=0;
51         Load_Store=0;
52         R0=0;
53         Mux_a_Reg=0;
54         #2 i_Reset=0; Load_Store=0; Mux_a_Reg=8'b00000000;
55         #2 Sel_RX=3'b000; Load_Store=1; Mux_a_Reg=8'b00000010;
56         #2 Sel_RX=3'b001; Load_Store=1; Mux_a_Reg=8'b00000011;
57         #2 Sel_RX=3'b010; Load_Store=1; Mux_a_Reg=8'b00000100;
58         #2 Sel_RX=3'b011; Load_Store=1; Mux_a_Reg=8'b00000101;
59         #2 Sel_RX=3'b100; Load_Store=1; Mux_a_Reg=8'b00000110;
60         #2 Sel_RX=3'b101; Load_Store=1; Mux_a_Reg=8'b00000111;
61         #2 Sel_RX=3'b110; Load_Store=1; Mux_a_Reg=8'b00001000;
62         #2 Sel_RX=3'b111; Load_Store=1; Mux_a_Reg=8'b00001001;
63
64         #2 Sel_RX=3'b011; Load_Store=0;
65         #2 Sel_RY=3'b111; Load_Store=0;
66     end
67     always
68     #1 i_Clk = !i_Clk;
69 endmodule
```

Código Tb_ALU

```
23 module Tb_ALU();
24
25     reg[7:0] RY;
26     reg[7:0] RX;
27     reg[3:0] Sel_ALU;
28     wire[7:0] R0;
29     wire[2:0] Flags;
30
31     ALU uut (
32         .RY(RY),
33         .RX(RX),
34         .Sel_ALU(Sel_ALU),
35         .R0(R0),
36         .Flags(Flags)
37     );
38
39     initial
40     begin
41         RY=0;
42         RX=0;
43         Sel_ALU=0;
44
45         #2
46         RY= 8'd4;
47         RX= 8'd4;
48         Sel_ALU=4'b0001;
49
50         #2
51         RY= 8'd4;
52         RX= 8'd5;
53         Sel_ALU=4'b0001;
54
55         #2
56         RY= 8'd255;
57         RX= 8'd1;
58         Sel_ALU=4'b0000;
59
60         #2
61         RY= 8'd4;
62         RX= 8'd3;
63         Sel_ALU=4'b0001;
64
65         #2
66         RY= 8'd10;
67         RX= 8'd5;
68         Sel_ALU=4'b0001;
69
70         #2
71         RY= 8'd2;
72         RX= 8'd1;
73         Sel_ALU=4'b0000;
74
75         end
```

Código Tb_Administrador_de_Salidas

```
22 module Tb_Administrador_de_Salidas();
23     reg [2:0] RY;
24     reg [7:0] RX;
25     reg [2:0] Num;
26     reg [1:0] Sel_Salidas;
27     wire [7:0] o_Dataout;
28     wire [7:0] o_Addressdata;
29     wire ReadWrite;
30
31     Administrador_de_salidas uut (
32         .RY(RY),
33         .RX(RX),
34         .Num(Num),
35         .Sel_Salidas(Sel_Salidas),
36         .o_Dataout(o_Dataout),
37         .o_Addressdata(o_Addressdata),
38         .ReadWrite(ReadWrite)
39     );
40     initial
41         begin
42
43             RY=0; // Inicialización de las entradas
44             RX=0;
45             Num=0;
46             Sel_Salidas=0;
47
48             #2
49             RY= 4'd4; //Se asigna valor a las entradas
50             RX= 4'd5;
51             Num= 4'd6;
52
53             #2 Sel_Salidas = 0; // Se actualiza el selector cada 2 tiempos
54             #2 Sel_Salidas = 1;
55             #2 Sel_Salidas = 2;
56             #2 Sel_Salidas = 3;
57
58             #2 Sel_Salidas = 0;
59             #2 Sel_Salidas = 1;
60             #2 Sel_Salidas = 2;
61             #2 Sel_Salidas = 3;
62             #2 Sel_Salidas = 0;
63
64             #2 Sel_Salidas = 1;
65             #2 Sel_Salidas = 2;
66             #2 Sel_Salidas = 3;
67
68     end
```


Código Tb_Micro

```
23 module Tb_Micro;
24     reg [7:0] i_DataInbus;
25     reg [8:0] i_Instruction;
26     reg i_Clk;
27     reg i_Reset;
28     wire [8:0] o_Addressinstruction_Bus;
29     wire [7:0] o_Dataout_Bus;
30     wire [7:0] o_Addressdata_Bus;
31     wire ReadWrite;
32
33     MicroUAZ uut(
34         .i_DataInbus(i_DataInbus),
35         .i_Instruction(i_Instruction),
36         .i_Clk(i_Clk),
37         .i_Reset(i_Reset),
38         .o_Addressinstruction_Bus(o_Addressinstruction_Bus),
39         .o_Dataout_Bus(o_Dataout_Bus),
40         .o_Addressdata_Bus(o_Addressdata_Bus),
41         .ReadWrite(ReadWrite)
42     );
43
44     initial
45         begin
46             i_DataInbus=0;
47             i_Instruction=0;
48             i_Reset=1;
49             i_Clk=0;
50             #2 i_Reset=0; i_DataInbus=8'b00000011; i_Instruction=9'b00000000;
51             #2 i_DataInbus=8'b10000011; i_Instruction=9'b001000111;
52             #2 i_DataInbus=8'b01000011; i_Instruction=9'b010001001;
53             #2 i_DataInbus=8'b00001111; i_Instruction=9'b011010010;
54             #2 i_DataInbus=8'b00000010; i_Instruction=9'b100011011;
55             #2 i_DataInbus=8'b00100001; i_Instruction=9'b101000000;
56             #2 i_DataInbus=8'b00010011; i_Instruction=9'b110011001;
57             #2 i_DataInbus=8'b00010011; i_Instruction=9'b111010110;
58             #2 i_DataInbus=8'b00000111; i_Instruction=9'b000000000;
59
60         end
61     initial
62         begin
63             i_Clk<=0;
64             i_Reset<=1;
65             #1 i_Reset<=0;
66             #1 i_Clk<=1;
67         end
68     always@(*)
69         begin
70             #1 i_Clk <= ~i_Clk;
71         end
72 endmodule
```

Código Tb_Microcontrolador

```
23 module Tb_Microcontrolador(  
24     );  
25     reg i_Clk;  
26     reg i_Reset;  
27     wire [7:0] o_Dataout_Bus;  
28  
29     Microcontrolador uut(  
30         .i_Clk(i_Clk),  
31         .i_Reset(i_Reset),  
32         .o_Dataout_Bus(o_Dataout_Bus)  
33     );  
34  
35     initial  
36     begin  
37         i_Clk<=0;  
38         i_Reset<=1;  
39         #1 i_Reset<=0;  
40         #1 i_Clk<=1;  
41     end  
42     always@(*)  
43     begin  
44         #1 i_Clk <= ~i_Clk; //clk invierte su valor  
45     end  
46 endmodule
```

Memorias de la multiplicación y la división

Memoria de instrucciones de la Multiplicación

1	001000000
2	001001111
3	001010111
4	001011001
5	001100000
6	010000101
7	101100000
8	110001000
9	101000100
10	001000000
11	001000001
12	110010001
13	101000010
14	111101011
15	111101010
16	100100111

Memoria de datos de la Multiplicación

1	00000110
2	00000001
3	00000100
4	00000011
5	00000100
6	00000101
7	00000110
8	00000111
9	00001000
10	00001001
11	00001010
12	00001011
13	00001100
14	00001101
15	00001110
16	00001111
17	00010000
18	00010001
19	00010010

Memoria de instrucciones de la División

1	001000000
2	001001010
3	001010100
4	001011001
5	001100000
6	010000101
7	010011110
8	101100000
9	110011000
10	101000100
11	101001000
12	110010001
13	101000010
14	111110011
15	111110010
16	111101110
17	111101111
18	100100111
19	00010010

Memoria de datos de la División

1	00010001
2	00000001
3	00000100
4	00000111
5	00000100
6	00000101
7	00000110
8	00000111
9	00001000
10	00001001
11	00001010
12	00001011
13	00001100
14	00001101
15	00001110
16	00001111
17	00010000
18	00010001