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 "A"

Reporte de MicroUAZ

RESUMEN:

Se elaboró un microcontrolador con una arquitectura tipo Harvard, el propósito del trabajo es implementarlo para hacer un algoritmo que realicé 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

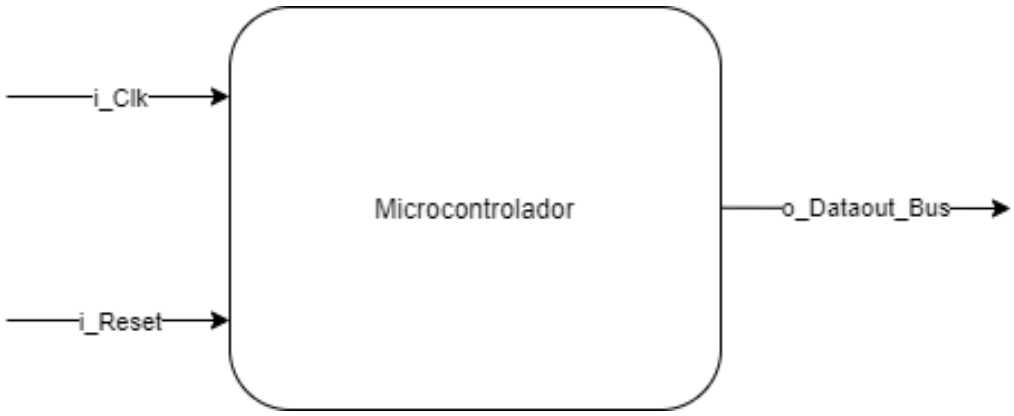

Figura 1. Caja negra de Microcotrolador, presentado sus entradas y salidas

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

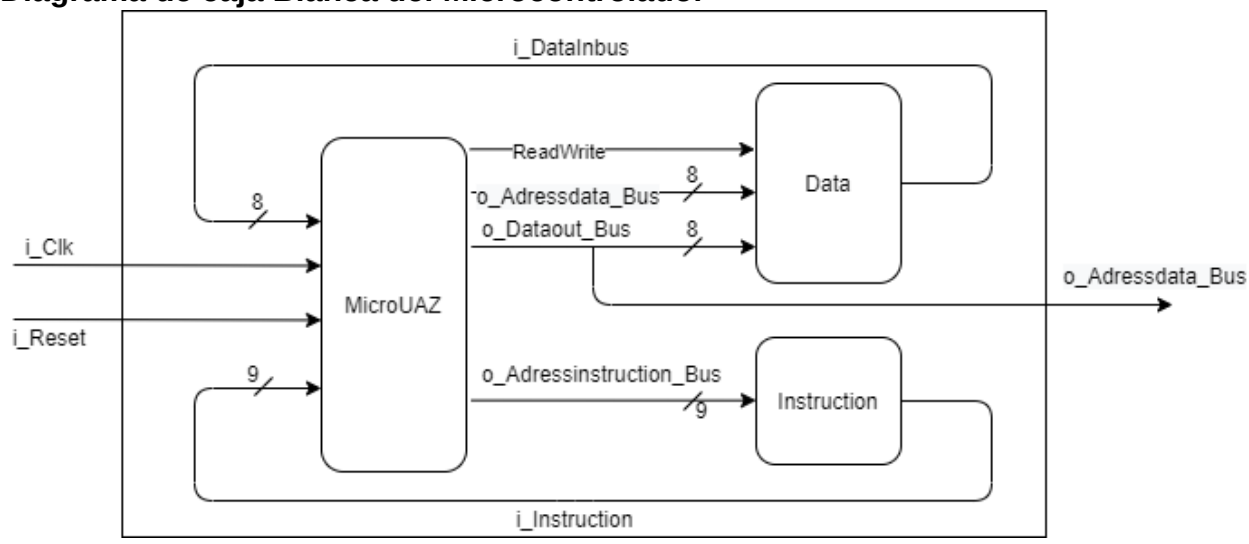

Figura 2. Caja blanca de Microcoontrolador, presentando los módulos internos

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

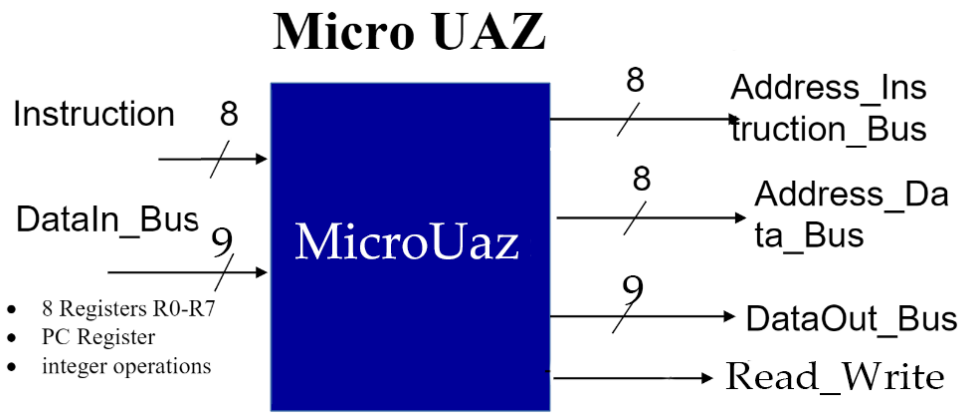

Figura 3. Caja negra de MicroUAZ, presentado sus entradas y salidas

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=~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 3. Tabla que muestra las señales de entrada y salida de la caja negra de MicroUAZ

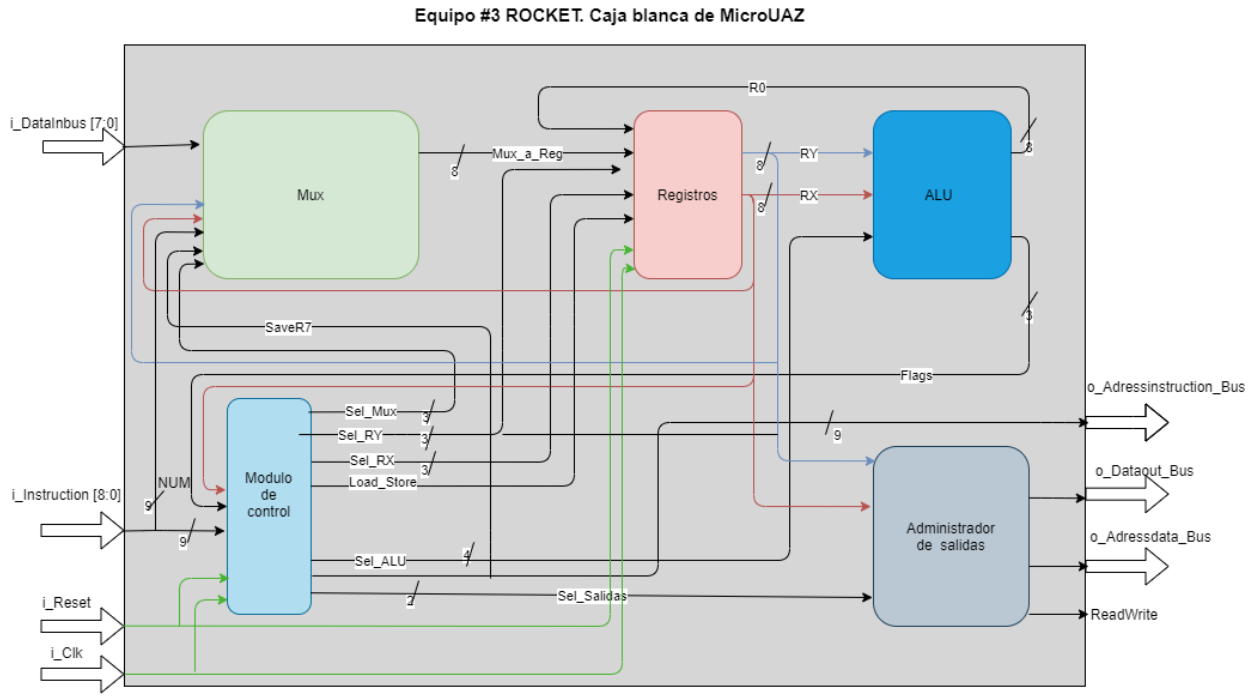

Figura 4. Caja negra de MicroUAZ, presentando sus módulos internos

Tabla de descripción

Modulo	Descripcion
Modulo de control	Se encarga de decodificar la entrada instruccion y de dividir las taresa 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

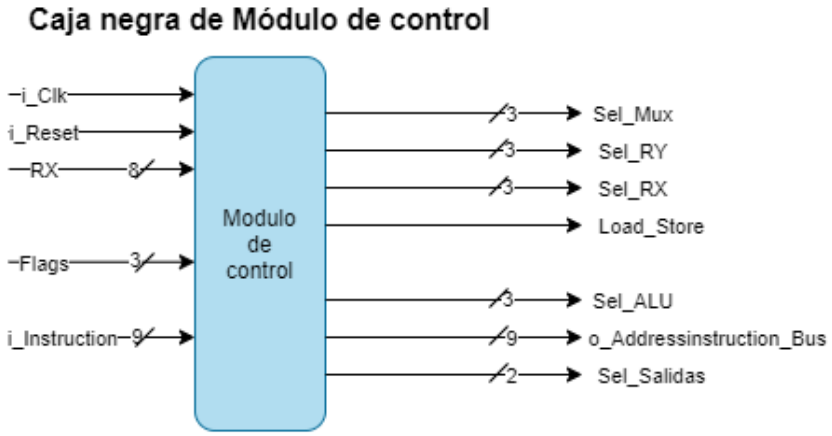

Figura 5. Caja negra de módulo de control, presentado sus entradas y salidas

Tabla de descripción

Nombre	Dirección	Num. de bits	Descripccion
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

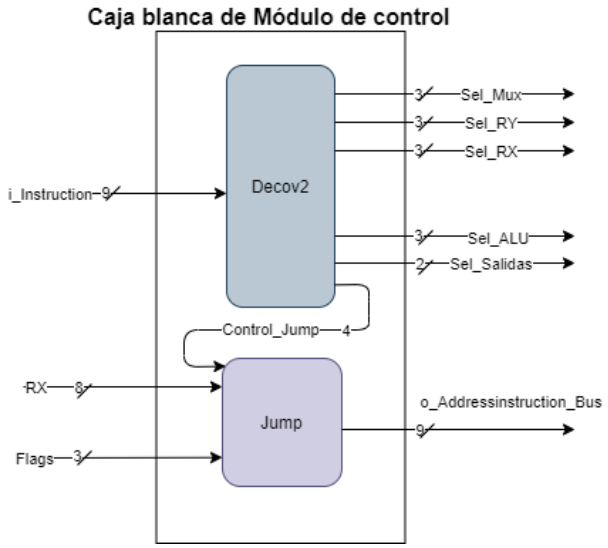

Figura 6. Caja blanca de módulo de control, presentando los módulos internos

Simulación de Decov2.v

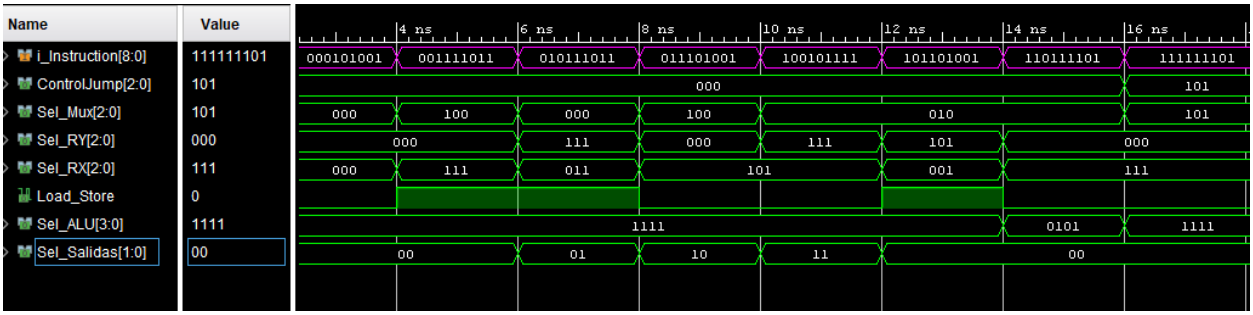

Figura 7. Simulación de Decov2

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

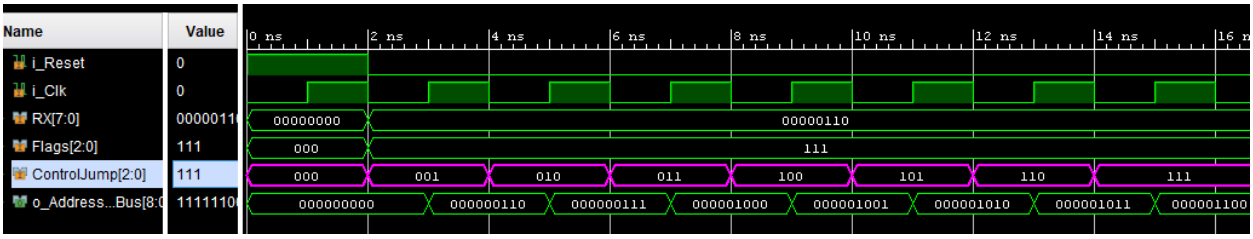

Figura 8. Simulación de Jump

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.

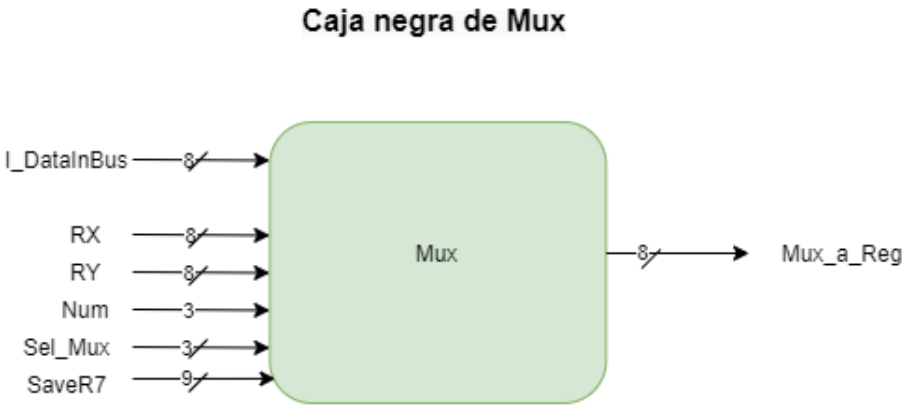

Figura 9. Caja negra del módulo Mux, presentado sus entradas y salidas

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

Descripcion 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

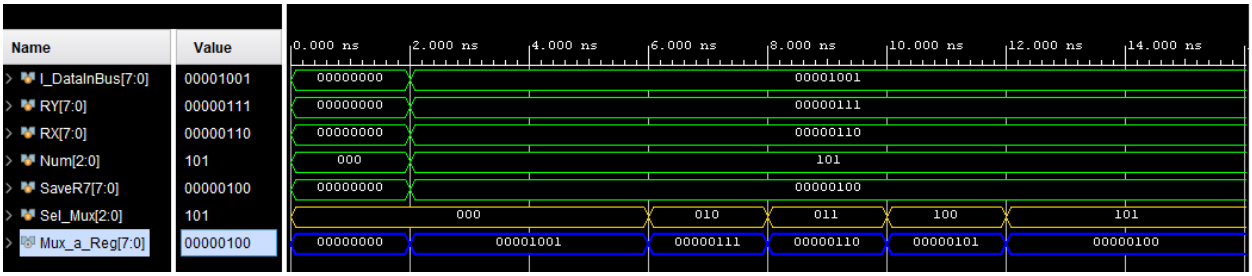

Figura 10. Simulación de Mux

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.

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

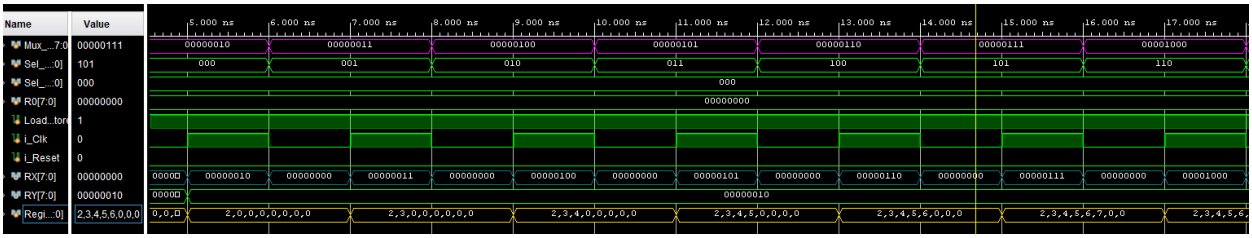

Figura 12. Simulación de Registros

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.

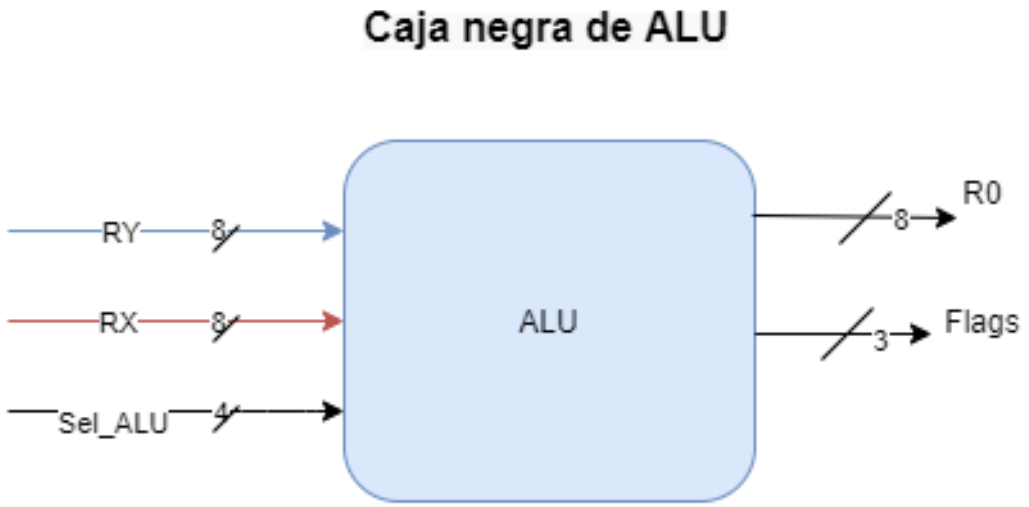

Figura 11. Caja negra del módulo ALU, presentado sus entradas y salidas

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)

Tabla 11. Descripción de señales del modulo ALU

Descripción funcional.

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 del la señal Sel_ALU como se indica a continuación:

Sel_ALU	R0
000	R0=RY+RX
001	R0=RY-RX
010	R0=RY<<RX
011	R0=RY>>RX
100	R0=~RX
101	R0=RY&RX
110	R0=RY|RX
111	R0=RY^RX

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

Simulación de ALU

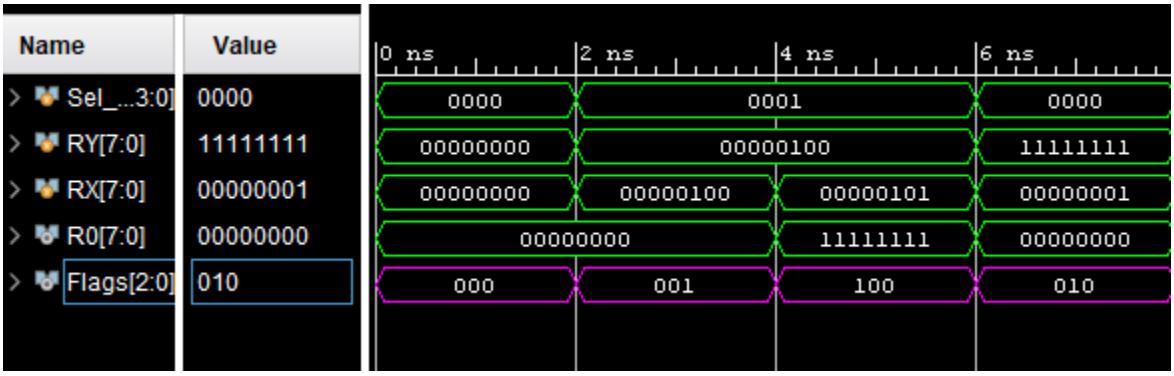

Figura 14. Simulación de ALU

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).

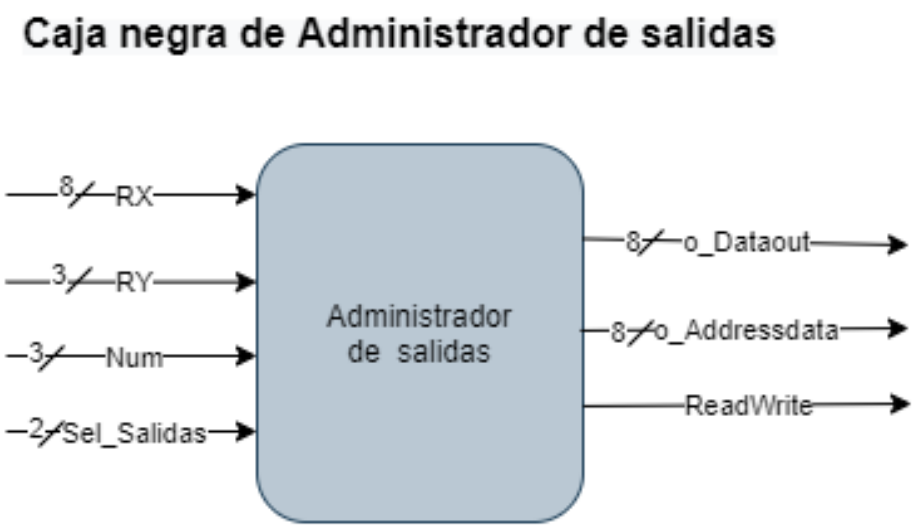

Figura 15. Caja negra del módulo ALU, presentado sus entradas y salidas

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

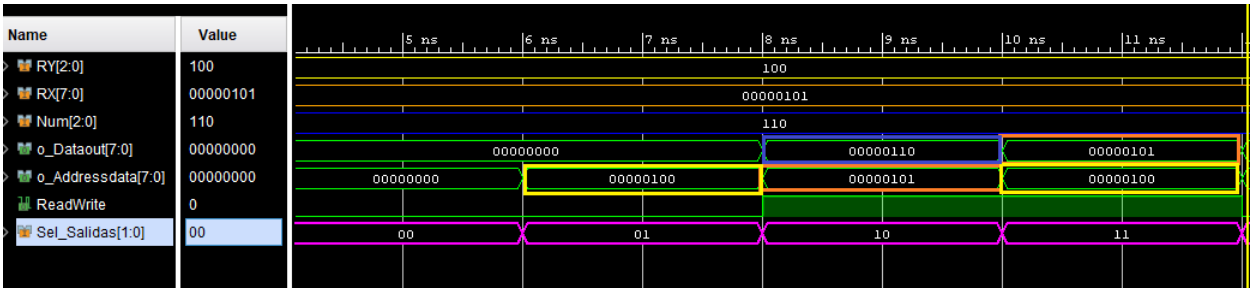

Figura 15. Simulación de Administrador_de_Salidas

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_Addresdata 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.

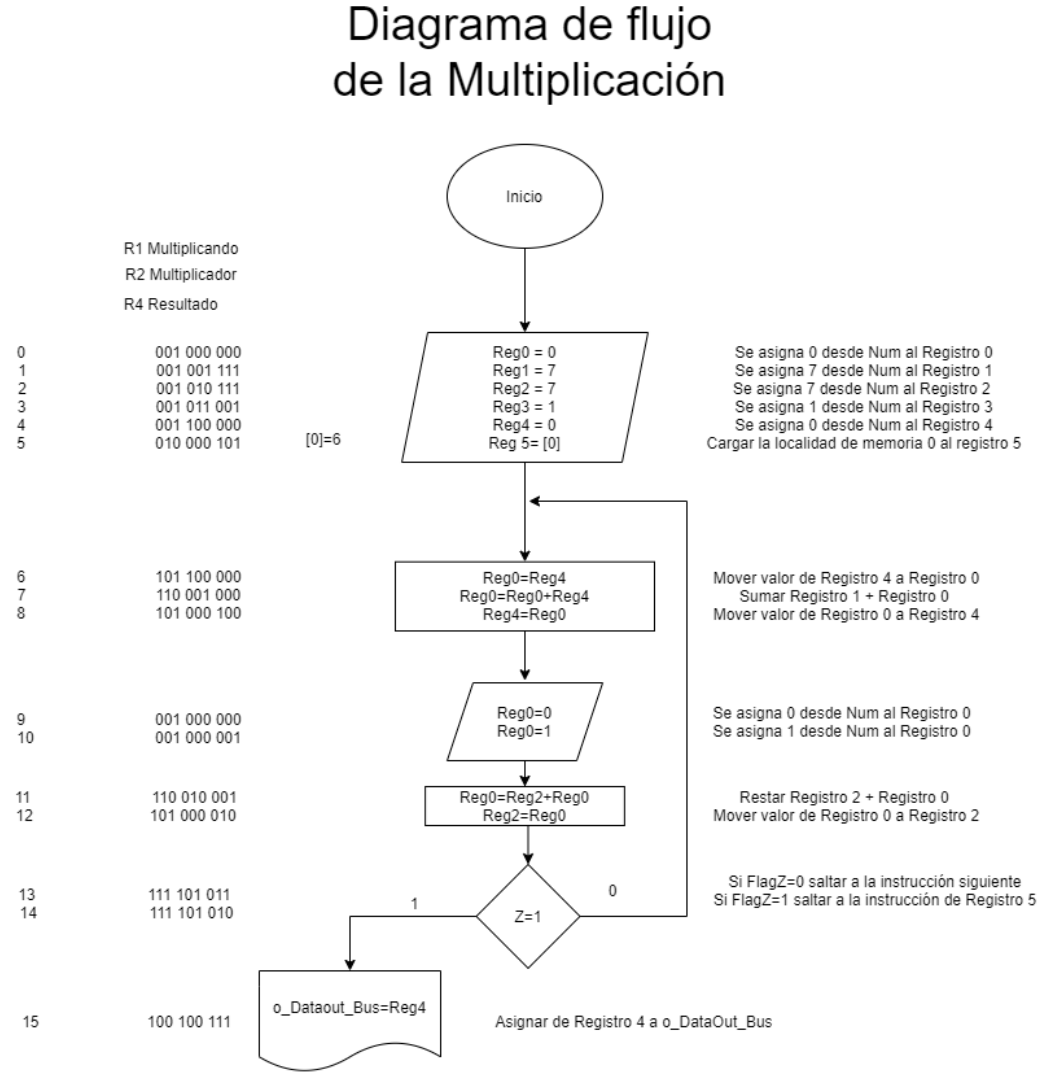

Figura 16. Diagrama de flujo de la multiplicación que muestra las instrucciones para realizar dicha operación.

Simulación de Multiplicación

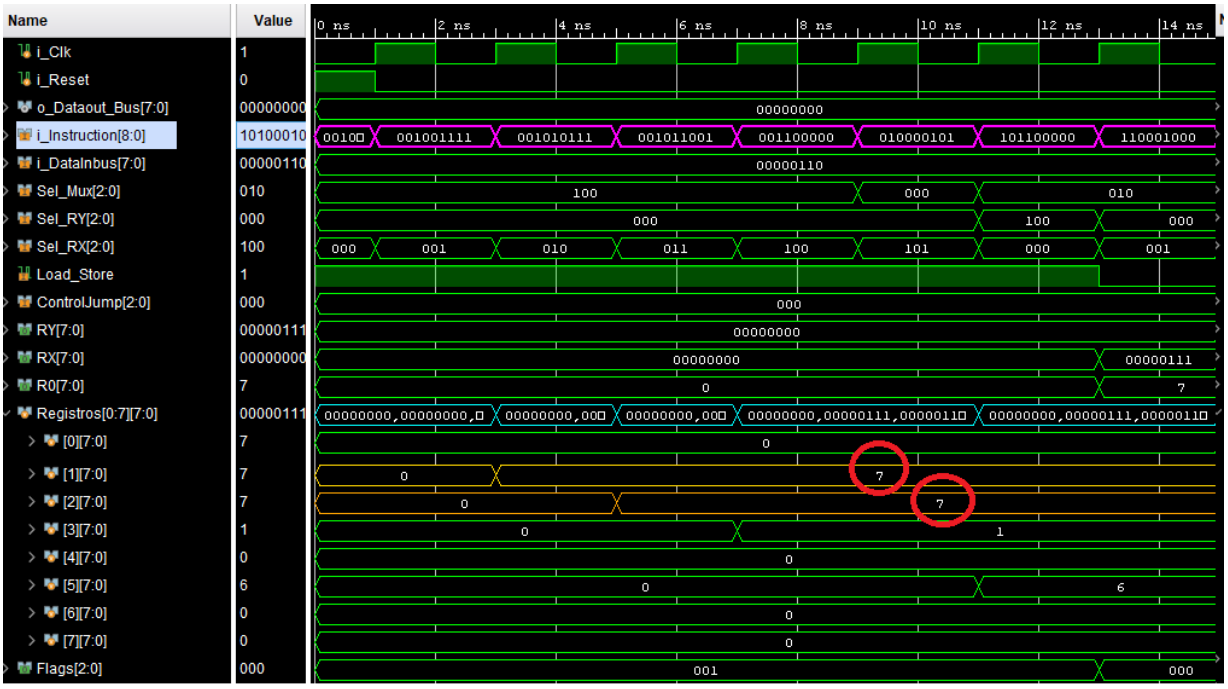

Figura 17. Simulación de Microcontrolador realizando la multiplicación, donde se muestran el multiplicador y el multiplicando.

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.

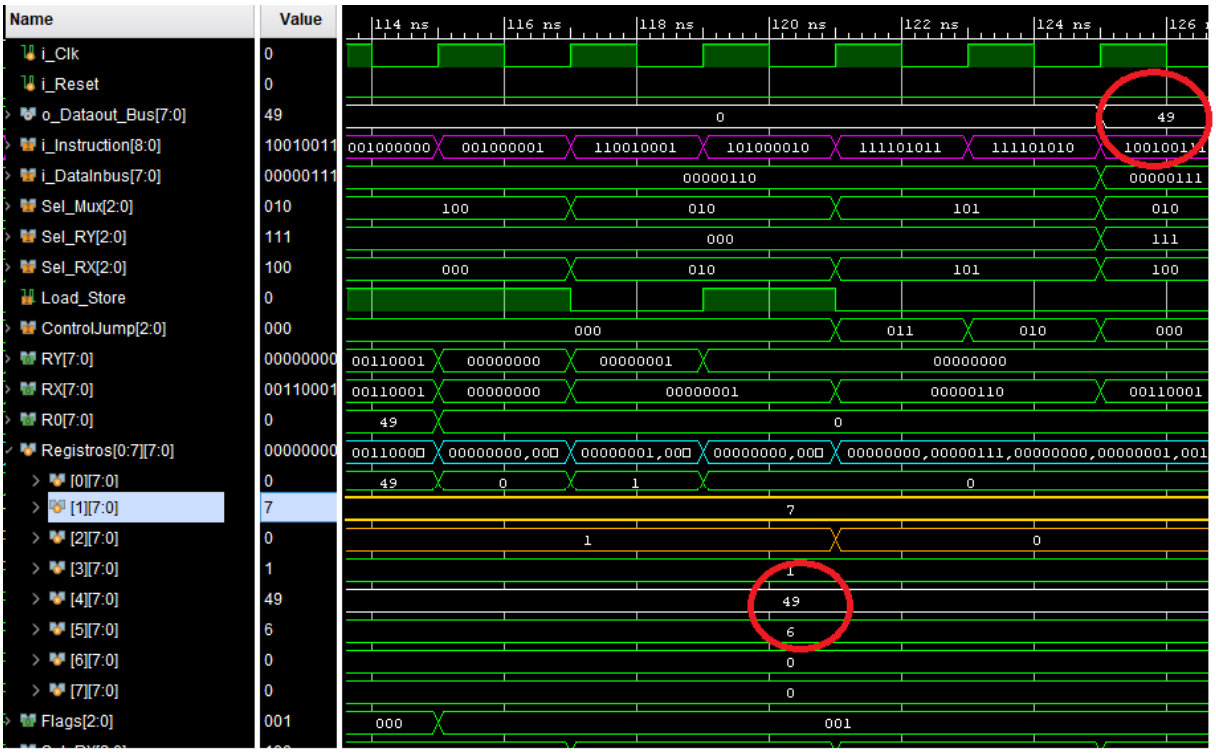

Figura 18. Simulación de Microcontrolador realizando la multiplicación, donde se muestra el producto de la multiplicación

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

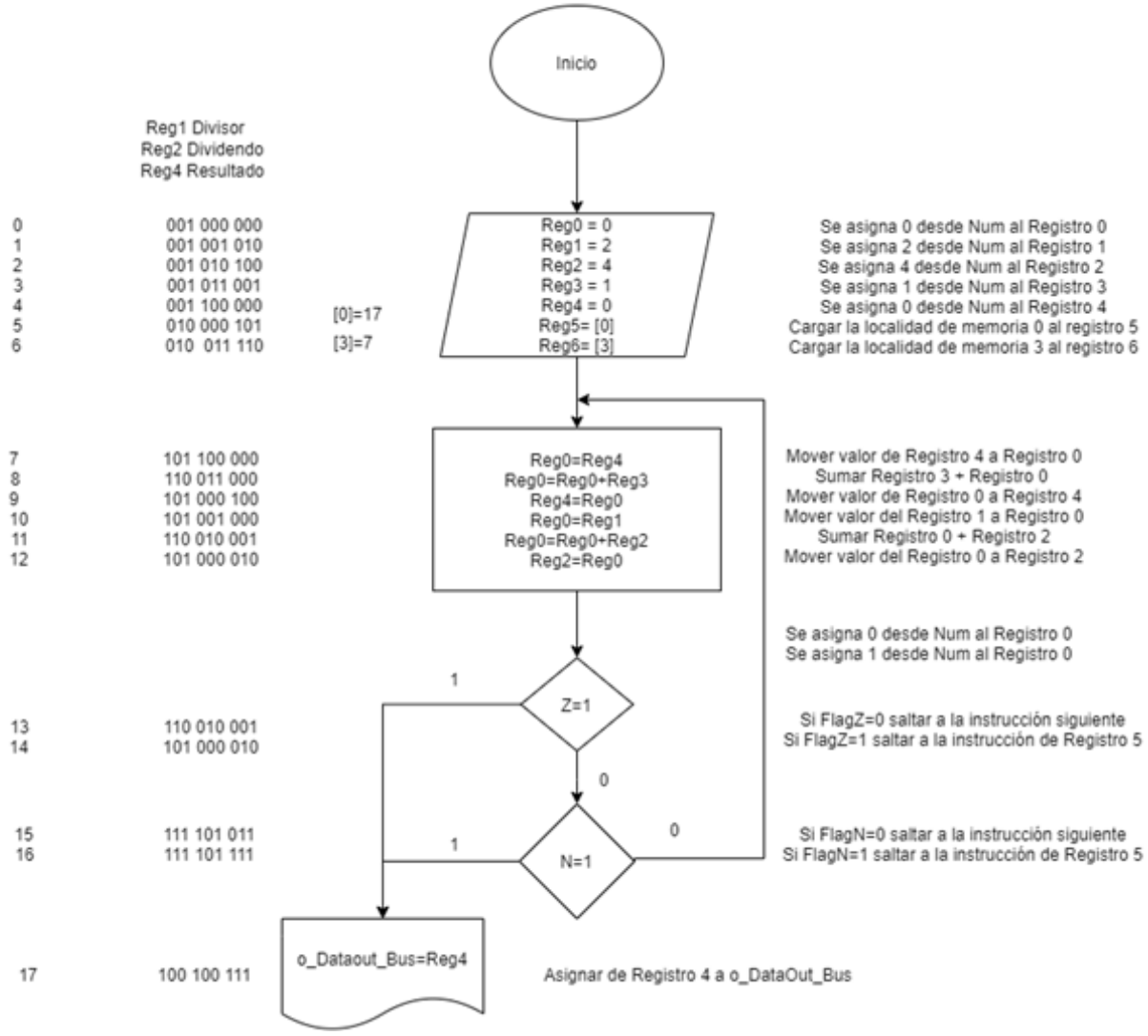

Figura 19. Diagrama de flujo de la división que muestra las instrucciones para realizar dicha operación.

Simulación de División

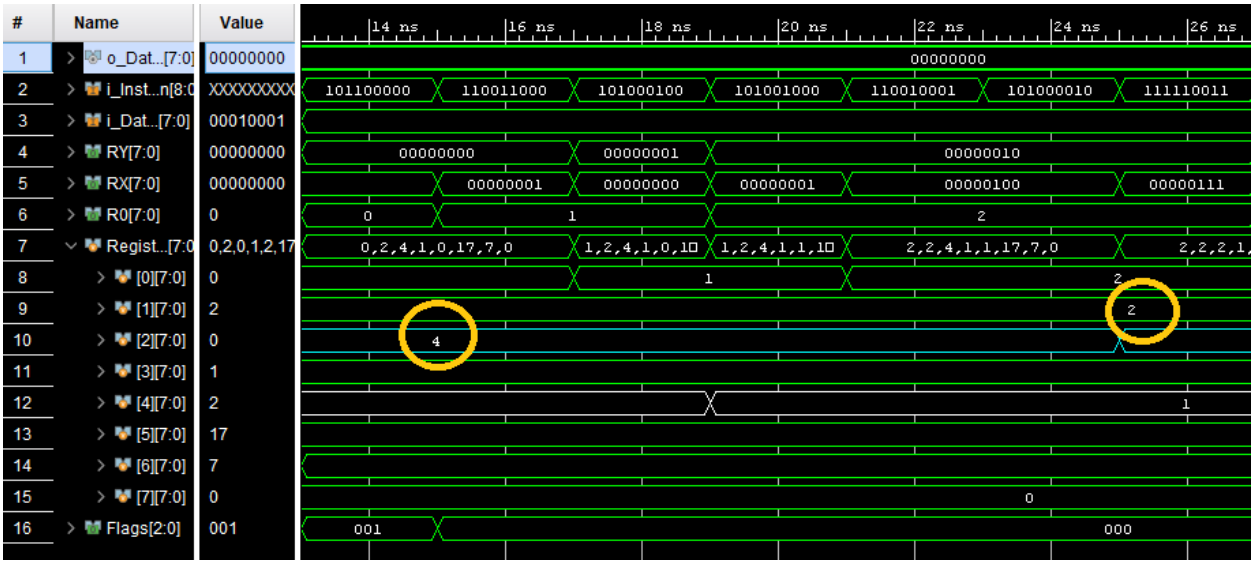

Figura 20. Simulación de Microcontrolador realizando la división, donde se muestran el divisor y el dividendo.

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.

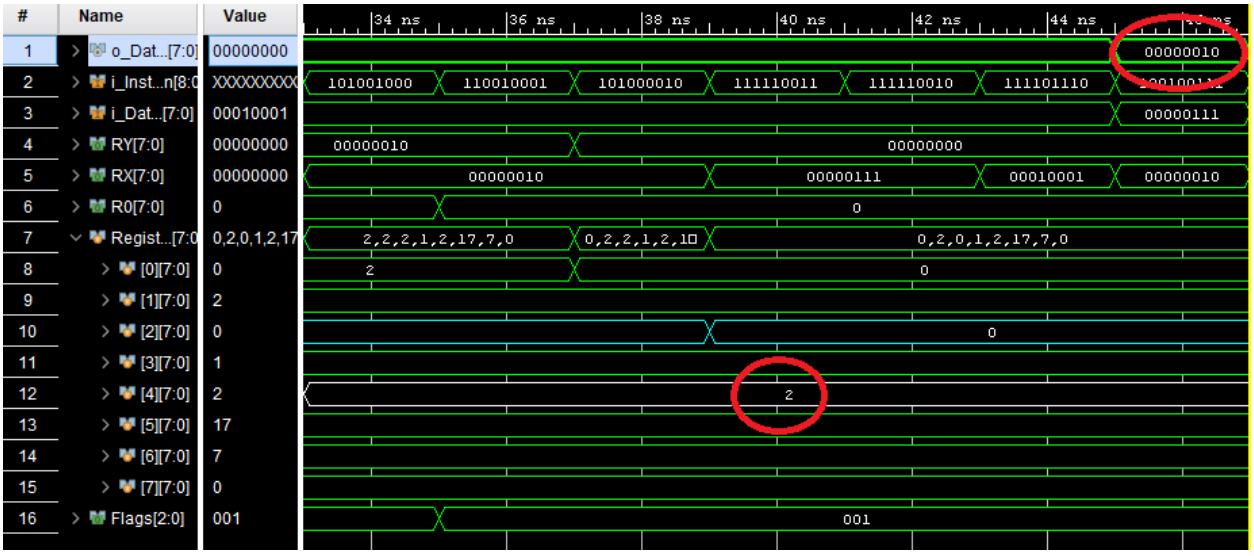

Figura 21. Simulación de Microcontrolador realizando la división, donde se muestra el cociente de la división

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 basto 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 resulto 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

```
module Decov2(
            input [8:0] i_Instruction,

            output reg [2:0] ControlJump,
            output reg [2:0] Sel_Mux,
            output reg [2:0] Sel_RY,
            output reg [2:0] Sel_RX,
            output reg Load_Store,
            output reg [3:0] Sel_ALU,
            output reg [1:0] Sel_Salidas
    );

              always@(i_Instruction)
              begin
                 case (i_Instruction[8:6])
                    3'b001:
                    begin
                             Sel_Mux=3'b100;
                             Sel_RY=3'b000;
                             Sel_RX=i_Instruction[5:3];
                             Load_Store=1'b1;
                             Sel_ALU=4'b1111;
                             Sel_Salidas=2'b00;
                             ControlJump=3'b000;

                             end
                    3'b010:
                    begin
                             Sel_Mux=3'b000;
                             Sel_RY=i_Instruction[5:3];
                             Sel_RX=i_Instruction[2:0];
                             Load_Store=1'b1;
                             Sel_ALU=4'b1111;
                             Sel_Salidas=2'b01;
                             ControlJump=3'b000;

                             end
                    3'b011:
                    begin
                              Sel_Mux=3'b100;
                              Sel_RY=3'b000;
                              Sel_RX=i_Instruction[5:3];
                              Load_Store=1'b0;
                              Sel_ALU=4'b1111;
                              Sel_Salidas=2'b10;
                              ControlJump=3'b000;
                              end
                    3'b100:
                    begin
                              Sel_Mux=3'b010;
                              Sel_RY=i_Instruction[2:0];
                              Sel_RX=i_Instruction[5:3];
                              Load_Store=1'b0;
                              Sel_ALU=4'b1111;
                              Sel_Salidas=2'b11;
                              ControlJump=3'b000;
                              end
                    3'b101:
                    begin
                             Sel_Mux=3'b010;
                             Sel_RY=i_Instruction[5:3];
                             Sel_RX=i_Instruction[2:0];
                             Load_Store=1'b1;
                             Sel_ALU=4'b1111;
                             Sel_Salidas=2'b00;
                             ControlJump=3'b000;
                             end
                    3'b110:
                    begin
                             Sel_Mux=3'b010;
                             Sel_RY=3'b000;
                             Sel_RX=i_Instruction[5:3];
                             Load_Store=1'b0;
                             Sel_ALU={1'b0,i_Instruction[2:0]};
                             Sel_Salidas=2'b00;
                             ControlJump=3'b000;

                             end
                    3'b111:
                    begin
                             Sel_Mux=3'b101;
                             Sel_RY=3'b000;
                             Sel_RX=i_Instruction[5:3];
                             Load_Store=1'b0;
                             Sel_ALU=4'b1111;
                             Sel_Salidas=2'b00;
                             ControlJump=i_Instruction[2:0];
                             end
                    default:
                    begin
                             Sel_Mux=0;
                             Sel_RY=0;
                             Sel_RX=0;
                             Load_Store=0;
                             Sel_ALU=4'b1111;
                             Sel_Salidas=0;
                             ControlJump=0;
                             end
                endcase
            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:
42                                   o_Addressinstruction_Bus=RX;
43                           3'b010:
44                               begin
45                                   if (Flags[0])
46                                       o_Addressinstruction_Bus=o_Addressinstruction_Bus+9'b000000001;
47                                   else
48                                       o_Addressinstruction_Bus=RX;
49                               end
50                           3'b011:
51                               begin
52                                   if (~Flags[0])
53                                      o_Addressinstruction_Bus=RX;
54                                   else
55                                       o_Addressinstruction_Bus=o_Addressinstruction_Bus+9'b000000001;
56                               end
57                           3'b100:
58                               begin
59                                   if (Flags[1])
60                                      o_Addressinstruction_Bus=o_Addressinstruction_Bus+9'b000000001;
61                                   else
62                                       o_Addressinstruction_Bus=RX;
63                               end
64                           3'b101:
65                               begin
66                                   if (~Flags[1])
67                                       o_Addressinstruction_Bus=RX;
68                                   else
69                                       o_Addressinstruction_Bus=o_Addressinstruction_Bus+9'b000000001;
70                               end
71
72                           3'b110:
73                               begin
74                                   if (Flags[2])
75                                   o_Addressinstruction_Bus=o_Addressinstruction_Bus+9'b000000001;
76                                   else
77                                   o_Addressinstruction_Bus=RX;
78
79                               end
80                           3'b111:
81                               begin
82                                   if (~Flags[2])
83                                   o_Addressinstruction_Bus=RX;
84                                   else
85                                   o_Addressinstruction_Bus=o_Addressinstruction_Bus+9'b000000001;
86
87
88                                   end
89                       endcase
90              end
91  endmodule
```

Código Modulo_de_Control

```
module Modulo_de_Control(
    input [7:0] RX,
    input [2:0] Flags ,
    input [8:0] i_Instruction,
    input i_Clk,
    input i_Reset,
    output [2:0] Sel_Mux,
    output [2:0] Sel_RY,
    output [2:0] Sel_RX,
    output Load_Store,
    output [2:0] Sel_Registros,
    output [3:0] Sel_ALU,
    output [8:0] o_Addressinstruction_Bus,
    output [1:0] Sel_Salidas
);

wire [2:0] ControlJump;

Jump Brinco(
   .ControlJump(ControlJump),
   .RX(RX),
   .Flags(Flags),
   .i_Clk(i_Clk),
   .i_Reset(i_Reset),
   .o_Addressinstruction_Bus(o_Addressinstruction_Bus)
);

Decov2 Decodificador(
    .i_Instruction(i_Instruction),
    .Sel_Mux(Sel_Mux),
    .Sel_RY(Sel_RY),
    .Sel_RX(Sel_RX),
    .Load_Store(Load_Store),
    .Sel_ALU(Sel_ALU),
    .Sel_Salidas(Sel_Salidas),
    .ControlJump(ControlJump)
);

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

```
module ALU #(parameter N=8) (
    input [7:0] RY,
    input [7:0] RX,
    input wire [3:0] Sel_ALU,
    output  [7:0] R0,
    output  [2:0] Flags
);

wire[N-1:0]salidasuma;
wire[N-1:0]salidaresta;
wire[N-1:0]salidaDizquierda;
wire[N-1:0]salidaDderecha;
wire[N-1:0]salidanot;
wire[N-1:0]salidaand;
wire[N-1:0]salidaor;
wire[N-1:0]salidaxor;
wire[N-1:0]salidanada;
wire[3:0] Suma_o_Resta;
wire c_out_suma;
wire c_out_resta;


SumadorNbits #(.N(N)) Op0(
    .a(RX),
    .b(RY),
    .c_in(1'b0),
    .sum(salidasuma),
    .c_out(c_out_suma)
);

Op_Resta #(.N(N))Op1(
    .A(RY),
    .B(RX),
    .R(salidaresta)
);

Desplazamiento_Izquierda#(.N(N)) Op2(
    .A(RX),
    .Y(salidaDizquierda)
);

Desplazamiento_Derecha#(.N(N)) Op3(
    .A(RY),
    .Y(salidaDderecha)
);

Op_Not#(.N(N)) Op4(
    .A(RX),
    .Y(salidanot)
);

Op_And#(.N(N)) Op5(
    .A(RX),
    .B(RY),
    .Y(salidaand)
);

Op_Or#(.N(N)) Op6(
    .A(RX),
    .B(RY),
    .Y(salidaor)
);
Nada#(.N(N)) Opeeeee(
    .A(RX),
    .Y(salidanada)
);
Op_Xor#(.N(N)) Op7(
    .A(RX),
    .B(RY),
    .Y(salidaxor)
);

Mux_ALU #(.N(N)) MuxALU(
    .salidasuma(salidasuma),
    .salidaresta(salidaresta),
    .salidaDizquierda(salidaDizquierda),
    .salidaDderecha(salidaDderecha),
    .salidanot(salidanot),
    .salidaand(salidaand),
    .salidaor (salidaor),
    .salidaxor(salidaxor),
    .Operador(Sel_ALU),
    .SalidaOp(R0),
    .OperadorSalida(Suma_o_Resta),
    .salidanada(salidanada)
);

Flag_Indicator #(.N(N)) Banderas(
    .A(RY),
    .B(RX),
    .SalidaOp(R0),
    .c_out_suma(c_out_suma),
    .Suma_o_Resta(Suma_o_Resta),
    .Flags (Flags)
);

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

```
module MicroUAZ(
    input [7:0] i_DataInbus,
    input [8:0] i_Instruction,
    input  i_Clk,
    input  i_Reset,
    output [8:0] o_Addressinstruction_Bus,
    output [7:0] o_Dataout_Bus,
    output [7:0] o_Addressdata_Bus,
    output ReadWrite
);

wire [2:0] Sel_Mux;
wire [2:0] Sel_RY;
wire [2:0] Sel_RX;
wire Load_Store;
wire [2:0] w_Sel_Registros;
wire [3:0] Sel_ALU;
wire [1:0] Sel_Salidas;
wire [7:0] Mux_a_Reg;
wire [7:0] R0;
wire [7:0] RY;
wire [7:0] RX;
wire [2:0] Flags;

Mux Multiplexor(
    .i_DataInBus(i_DataInbus),
    .RY(RY),
    .RX(RX),
    .Num(i_Instruction),
    .SaveR7(o_Addressinstruction_Bus),
    .Sel_Mux(Sel_Mux),
    .Mux_a_Reg(Mux_a_Reg)
);

Modulo_de_Control Control(
    .RX(RX),
    .Flags(Flags),
    .i_Instruction(i_Instruction),
    .i_Clk(i_Clk),
    .i_Reset(i_Reset),
    .Sel_Mux(Sel_Mux),
    .Sel_RY(Sel_RY),
    .Sel_RX(Sel_RX),
    .Load_Store(Load_Store),
    .Sel_ALU(Sel_ALU),
    .o_Addressinstruction_Bus(o_Addressinstruction_Bus),
    .Sel_Salidas(Sel_Salidas)
);

Registros Banco_de_Registros(
    .R0(R0),
    .Mux_a_Reg(Mux_a_Reg),
    .Sel_RY(Sel_RY),
    .Sel_RX(Sel_RX),
    .Load_Store(Load_Store),
    .i_Clk(i_Clk),
    .i_Reset(i_Reset),
    .RY(RY),
    .RX(RX)
);

ALU UnidadAL(
    .RY(RX),
    .RX(RY),
    .Sel_ALU(Sel_ALU),
    .R0(R0),
    .Flags(Flags)
);

Administrador_de_salidas Salidas(
    .RY(Sel_RY),
    .RX(RX),
    .Num(i_Instruction),
    .Sel_Salidas(Sel_Salidas),
    .o_Dataout(o_Dataout_Bus),
    .o_Addressdata(o_Addressdata_Bus),
    .ReadWrite(ReadWrite)
);

endmodule
```

Código Data

```
module Data(
    input [7:0] i_DataInbus,
    input [7:0] o_Dataout,
    input ReadWrite,
    output reg [7:0] o_Addressdata_Bus
    );
    reg [7:0] mem [0:255];

    initial
        begin
            $readmemb("Data.mem",mem);
        end
    always@(i_DataInbus,o_Dataout,ReadWrite)
    begin
        if(ReadWrite)
            begin
                mem[i_DataInbus]<=o_Dataout;
                o_Addressdata_Bus<=mem[i_DataInbus];
            end
        else
            o_Addressdata_Bus<=mem[i_DataInbus];
    end
endmodule
```

Código Instruction

```
module Instruction(
input [8:0] i_Instruction,
output reg [8:0] o_Addressinstruction_Bus
);
reg [8:0] mem [0:255];

initial
$readmemb("Instruction.mem",mem,0,255);
always@(i_Instruction)
o_Addressinstruction_Bus <= mem[i_Instruction];

endmodule
```

Código Microcontrolador

```
module Microcontrolador(
    input  i_Clk,
    input  i_Reset,
    output [7:0] o_Dataout_Bus
    );
    wire [7:0] i_DataInbus;
    wire [8:0] i_Instruction;
    wire [8:0] o_Addressinstruction_Bus;
    wire [7:0] o_Addressdata_Bus;
    wire ReadWrite;

MicroUAZ Micro(
   .i_DataInbus(i_DataInbus),
   .i_Instruction(i_Instruction),
   .i_Clk(i_Clk),
   .i_Reset(i_Reset),
   .o_Addressinstruction_Bus(o_Addressinstruction_Bus),
   .o_Dataout_Bus(o_Dataout_Bus),
   .o_Addressdata_Bus(o_Addressdata_Bus),
   .ReadWrite(ReadWrite)
);

Data Ram(
    .i_DataInbus(o_Addressdata_Bus),
    .o_Dataout(o_Dataout_Bus),
    .o_Addressdata_Bus(i_DataInbus),
    .ReadWrite(ReadWrite)
);

Instruction Rom(
    .i_Instruction(o_Addressinstruction_Bus),
    .o_Addressinstruction_Bus(i_Instruction)
);

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;
56              #2
57              RX=6;
58              Flags=3'b111;
59              ControlJump=3'b010;
60
61              #2
62              RX=6;
63              Flags=3'b111;
64              ControlJump=3'b011;
65
66              #2
67              RX=6;
68              Flags=3'b111;
69              ControlJump=3'b100;
70
71              #2
72              RX=6;
73              Flags=3'b111;
74              ControlJump=3'b101;
75
76              #2
77              RX=6;
78              Flags=3'b111;
79              ControlJump=3'b110;
80
81              #2
82              RX=6;
83              Flags=3'b111;
84              ControlJump=3'b111;
85
86          end
87      always
88                      #1 i_Clk = !i_Clk;
89      endmodule
```

Código Tb_Mux

```
module Tb_Mux;
    reg [7:0] I_DataInBus;
    reg [7:0] RY;
    reg [7:0] RX;
    reg [2:0] Num;
    reg [7:0]SaveR7;
    reg [2:0] Sel_Mux;
    wire [7:0] Mux_a_Reg;

    Mux uut(
        .i_DataInBus(I_DataInBus),
        .RY(RY),
        .RX(RX),
        .Num(Num),
        .SaveR7(SaveR7),
        .Sel_Mux(Sel_Mux),
        .Mux_a_Reg(Mux_a_Reg)
    );
    initial
        begin
            I_DataInBus=0;
            RY=0;
            RX=0;
            Num=0;
            SaveR7=0;
            Sel_Mux=3'b000;
            #2 I_DataInBus=8'b00001001; RY=8'b00000111;
            RX=8'b00000110; Num=8'b00000101; SaveR7=8'b00000100;
            #2 Sel_Mux=3'b000;
            #2 Sel_Mux=3'b010;
            #2 Sel_Mux=3'b011;
            #2 Sel_Mux=3'b100;
            #2 Sel_Mux=3'b101;
        end
endmodule
```

Código Tb_Registros

```
module Tb_Registros;
    reg [7:0] Mux_a_Reg;
    reg [2:0] Sel_RX;
    reg [2:0] Sel_RY;
    reg [7:0] R0;
    reg Load_Store;
    reg i_Clk;
    reg i_Reset;
    wire [7:0] RX;
    wire [7:0] RY;

    Registros uut(
    .Mux_a_Reg(Mux_a_Reg),
    .Sel_RX(Sel_RX),
    .Sel_RY(Sel_RY),
    .R0(R0),
    .Load_Store(Load_Store),
    .i_Clk(i_Clk),
    .i_Reset(i_Reset),
    .RX(RX),
    .RY(RY)
    );

    initial
        begin
            i_Reset=1;
            i_Clk=0;
            Sel_RX=0;
            Sel_RY=0;
            Load_Store=0;
            R0=0;
            Mux_a_Reg=0;
            #2 i_Reset=0; Load_Store=0; Mux_a_Reg=8'b00000000;
            #2 Sel_RX=3'b000; Load_Store=1; Mux_a_Reg=8'b00000010;
            #2 Sel_RX=3'b001; Load_Store=1; Mux_a_Reg=8'b00000011;
            #2 Sel_RX=3'b010; Load_Store=1; Mux_a_Reg=8'b00000100;
            #2 Sel_RX=3'b011; Load_Store=1; Mux_a_Reg=8'b00000101;
            #2 Sel_RX=3'b100; Load_Store=1; Mux_a_Reg=8'b00000110;
            #2 Sel_RX=3'b101; Load_Store=1; Mux_a_Reg=8'b00000111;
            #2 Sel_RX=3'b110; Load_Store=1; Mux_a_Reg=8'b00001000;
            #2 Sel_RX=3'b111; Load_Store=1; Mux_a_Reg=8'b00001001;

            #2 Sel_RX=3'b011; Load_Store=0;
            #2 Sel_RY=3'b111; Load_Store=0;
        end
        always
            #1 i_Clk = !i_Clk;
endmodule
```

Código Tb_ALU

```
module Tb_ALU();

    reg[7:0] RY;
    reg[7:0] RX;
    reg[3:0] Sel_ALU;
    wire[7:0] R0;
    wire[2:0] Flags;

    ALU uut (
        .RY(RY),
        .RX(RX),
        .Sel_ALU(Sel_ALU),
        .R0(R0),
        .Flags(Flags)
    );

        initial
        begin
            RY=0;
            RX=0;
            Sel_ALU=0;

            #2
            RY= 8'd4;
            RX= 8'd4;
            Sel_ALU=4'b0001;
            #2
            RY= 8'd4;
            RX= 8'd5;
            Sel_ALU=4'b0001;
            #2
            RY= 8'd255;
            RX= 8'd1;
            Sel_ALU=4'b0000;

            #2
            RY= 8'd4;
            RX= 8'd3;
            Sel_ALU=4'b0001;
            #2
            RY= 8'd10;
            RX= 8'd5;
            Sel_ALU=4'b0001;
            #2
            RY= 8'd2;
            RX= 8'd1;
            Sel_ALU=4'b0000;

        end
```

Código Tb_Administrador_de_Salidas

```
module Tb_Administrador_de_Salidas();
    reg [2:0] RY;
    reg [7:0] RX;
    reg [2:0] Num;
    reg [1:0] Sel_Salidas;
    wire [7:0] o_Dataout;
    wire [7:0] o_Addressdata;
    wire ReadWrite;

    Administrador_de_salidas uut (
        .RY(RY),
        .RX(RX),
        .Num(Num),
        .Sel_Salidas(Sel_Salidas),
        .o_Dataout(o_Dataout),
        .o_Addressdata(o_Addressdata),
        .ReadWrite(ReadWrite)
    );
    initial
        begin

            RY=0;// Inicialización de las entradas
            RX=0;
            Num=0;
            Sel_Salidas=0;

            #2
            RY= 4'd4; //Se asigna valor a las entradas
            RX= 4'd5;
            Num= 4'd6;

            #2 Sel_Salidas = 0; // Se actualiza el selector cada 2 tiempos
            #2 Sel_Salidas = 1;
            #2 Sel_Salidas = 2;
            #2 Sel_Salidas = 3;

            #2 Sel_Salidas = 0;
            #2 Sel_Salidas = 1;
            #2 Sel_Salidas = 2;
            #2 Sel_Salidas = 3;
            #2 Sel_Salidas = 0;

            #2 Sel_Salidas = 1;
            #2 Sel_Salidas = 2;
            #2 Sel_Salidas = 3;

        end
```

Código Tb_Micro

```
module Tb_Micro;
    reg [7:0] i_DataInbus;
    reg [8:0] i_Instruction;
    reg  i_Clk;
    reg  i_Reset;
    wire [8:0] o_Addressinstruction_Bus;
    wire [7:0] o_Dataout_Bus;
    wire [7:0] o_Addressdata_Bus;
    wire ReadWrite;

    MicroUAZ uut(
        .i_DataInbus(i_DataInbus),
        .i_Instruction(i_Instruction),
        .i_Clk(i_Clk),
        .i_Reset(i_Reset),
        .o_Addressinstruction_Bus(o_Addressinstruction_Bus),
        .o_Dataout_Bus(o_Dataout_Bus),
        .o_Addressdata_Bus(o_Addressdata_Bus),
        .ReadWrite(ReadWrite)
    );

    initial
        begin
            i_DataInbus=0;
            i_Instruction=0;
            i_Reset=1;
            i_Clk=0;
            #2 i_Reset=0;i_DataInbus=8'b00000011; i_Instruction=9'b000000000;
            #2 i_DataInbus=8'b10000011; i_Instruction=9'b001000111;
            #2 i_DataInbus=8'b01000011; i_Instruction=9'b010001001;
            #2 i_DataInbus=8'b00001111; i_Instruction=9'b011010010;
            #2 i_DataInbus=8'b00000010; i_Instruction=9'b100011011;
            #2 i_DataInbus=8'b00100001; i_Instruction=9'b101000000;
            #2 i_DataInbus=8'b00010011; i_Instruction=9'b110011001;
            #2 i_DataInbus=8'b00010011; i_Instruction=9'b111010110;
            #2 i_DataInbus=8'b00000111; i_Instruction=9'b000000000;

        end
initial
    begin
        i_Clk<=0;
        i_Reset<=1;
        #1 i_Reset<=0;
        #1 i_Clk<=1;
    end
    always@(*)
    begin
        #1 i_Clk <= ~i_Clk;
    end
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

```
001000000
001001111
001010111
001011001
001100000
010000101
101100000
110001000
101000100
001000000
001000001
110010001
101000010
111101011
111101010
100100111
```

Memoria de datos de la Multiplicación

```
00000110
00000001
00000100
00000011
00000100
00000101
00000110
00000111
00001000
00001001
00001010
00001011
00001100
00001101
00001110
00001111
00010000
00010001
00010010
```

Memoria de instrucciones de la División

```
001000000
001001010
001010100
001011001
001100000
010000101
010011110
101100000
110011000
101000100
101001000
110010001
101000010
111110011
111110010
111101110
111101111
100100111
00010010
```

Memoria de datos de la División

```
00010001
00000001
00000100
00000111
00000100
00000101
00000110
00000111
00001000
00001001
00001010
00001011
00001100
00001101
00001110
00001111
00010000
00010001
```