BENEMÉRITA UNIVERSIDAD
AUTÓNOMA DE ZACATECAS
"Francisco García Salinas"

UNIDAD ACADÉMICA DE INGENIERÍA ELÉCTRICA

PROGRAMA ACADÉMICO: INGENIERIA EN ROBÓTICA Y MECATRÓNICA

SISTEMAS DIGITALES III

MICROCONTROLADOR

ALUMNOS: **-José Roberto Novoa López**
-Julio Ángel Pérez Dávila

DOCENTE: **Dr. Remberto Sandoval Arrechiga**

GRUPO: **7–"B"**

28/11/2020

Resumen

El Microcontrolador es un circuito integrado que es el componente principal de una aplicación embebida. Es como una pequeña computadora que incluye sistemas para controlar elementos de entrada/salida. También incluye a un procesador y por supuesto memoria que puede guardar el programa y sus variables (flash y RAM). Funciona como una mini PC. Su función es la de automatizar procesos y procesar información

Por lo tanto, realizares un microcontrolador RISC Harvard de 8 bits con característica almacenamiento de instrucciones de rápida ejecución.

Índice

Introducción 4

Tipos de Memorias 5

Requerimientos 6
Arquitectura 7
Caja blanca microcontrolador 7
Num 8
Caja negra *8*
Descripción funcional *8*
Tabla de verdad *9*
Unidad de control 9
Caja negra *9*
Descripción funcional *10*
Tabla de verdad *10*
Deco 11
Caja negra *11*
Descripción funcional *11*
Jump *12*
Selector 12
Caja negra *12*
Descripción funcional *13*
Tabla de verdad *13*
Alu 14
Caja negra *14*
Descripción funcional *14*
Tabla de verdad *15*
Banderas *15*
Banco de registros 16
Caja negra *16*
Descripción funcional *16*
Tabla de verdad *17*
Bus 17
Caja negra *17*
Descripción funcional *18*
Tabla de verdad *18*
Implementación 18

Pruebas 19
Selector 19
Bus 19
Num 20
Banco de registros 21
Alu 21
Jump 22
Micro8bits_Harvard 22
Compu_personal 23
Análisis de resultados 23
Conclusiones 26
Referencias 26
Código de implementación 27
Selector 31
Bus 32
Num 34
Banco de registros 35
Alu 36
Control 37
Deco 39
Jump 41
Test benches 44
Compu_personal 44
Micro8bits_Harvard 44
Selector 46
Bus 48
Num 50
Banco de registros 51
Alu 53
Control 55

Introducción

.

El microcontrolador se aplica en toda clase de inventos y productos donde se requiere seguir un proceso automático dependiendo de las condiciones de distintas entradas.

Elementos de un Microcontrolador

- Microprocesador.
- Periféricos (entrada/salida).
- Memoria.

Como se compone un microcontrolador internamente

Un procesador incluye al menos tres elementos, ALU, unidad de control y registros.

- **ALU** (Unidad Aritmética y Lógica). Está unidad está compuesta por los circuitos electrónicos digitales del tipo combinatorios (compuertas, sumadores, multiplicadores), cuya principal función es el realizar operaciones. Estas operaciones están divididas en tres tipos:
  - **Lógicas**. Como las operaciones básicas de las compuertas lógicas, como la suma lógica (OR), multiplicación lógica (AND), diferencia lógica (XOR) y negación (NOT). Una operación lógica sólo puede tener como entradas y como salidas una respuesta lógica (0 o 1). Esto dependiendo de los niveles de voltajes de una señal digital.
  - **Aritméticas**. Las operaciones aritméticas son la suma, resta, multiplicación y división. Dependiendo del procesador (8, 16, 32 o 64 bits) será la rapidez con la que se pueden hacer dichas operaciones.
  - **Misceláneas**. En estas operaciones caen todas las demás operaciones como la transferencia de bits (<< >>).
- **Unidad de control**. La unidad de control es el conjunto de sistemas digitales secuenciales (aquellos que tienen memoria) que permiten distribuir la lógica de las señales.

- **Registros**. Los registros son las memorias principales de los procesadores, ya que funcionan a la misma velocidad que el procesador a diferencia de otras memorias un tanto más lentas (como la RAM, FLASH o la CACHE). Los registros están construidos por Flip-Flops. Los Flip-Flops son circuitos digitales secuenciales.

Periféricos.

- Los periféricos son los circuitos digitales que nos permiten una interacción con el mundo «exterior» al microcontrolador. Su función es la de poder habilitar o deshabilitar las salidas digitales, leer sensores analógicos, comunicación con terminales digitales o sacar señales analógicas de una conversión digital.

    - **Puertos de entrada/salida** paralelos. Los puertos están relacionados al tamaño del procesador, es decir que un puerto de 8 bits es porque el procesador es de 8 bits. Un procesador de 64 bits, tiene la capacidad de tener un puerto de 64 bits.

    - **Puertos seriales**. Nos permiten transformar la información digital paralela (bytes de información) en tramas que se pueden transferir por una o varias líneas de comunicación. Existen por ejemplo: puerto serial, i2c, SPI, USB, CAN, etc.

    - **Periféricos analógicos**. Como los que convierten señales analógicas a digitales (ADC) o señales digitales a analógicas (DAC) o comparadores analógicos.

Tipos de Memorias

La memoria está dividida en tres:

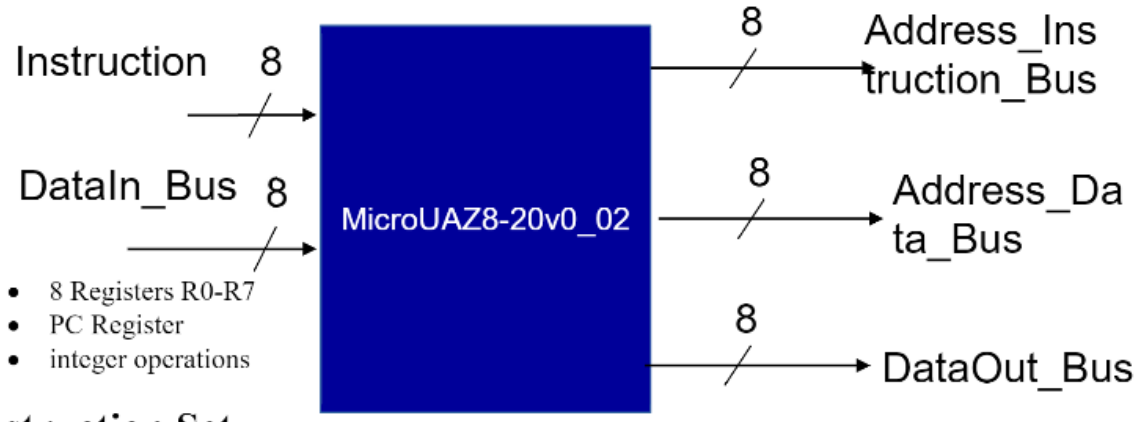
- La memoria para el programa (FLASH),
- la memoria para los datos o variables del programa (RAM)
- la memoria para configuraciones o no volátil (EEPROM)

Microcontrolador con Registros:

Los registros son las memorias digitales más rápidas. Hechas con Flip-Flops y generalmente funcionan a una velocidad cercana a la del procesador. En algunos procesadores, también incluyen un tipo de memoria llamada CACHE. Está no puede guardar operaciones y sólo es un puente entre el procesador y la memoria principal.

Requerimientos

Micro UAZ 8Bits 2020 v0.02

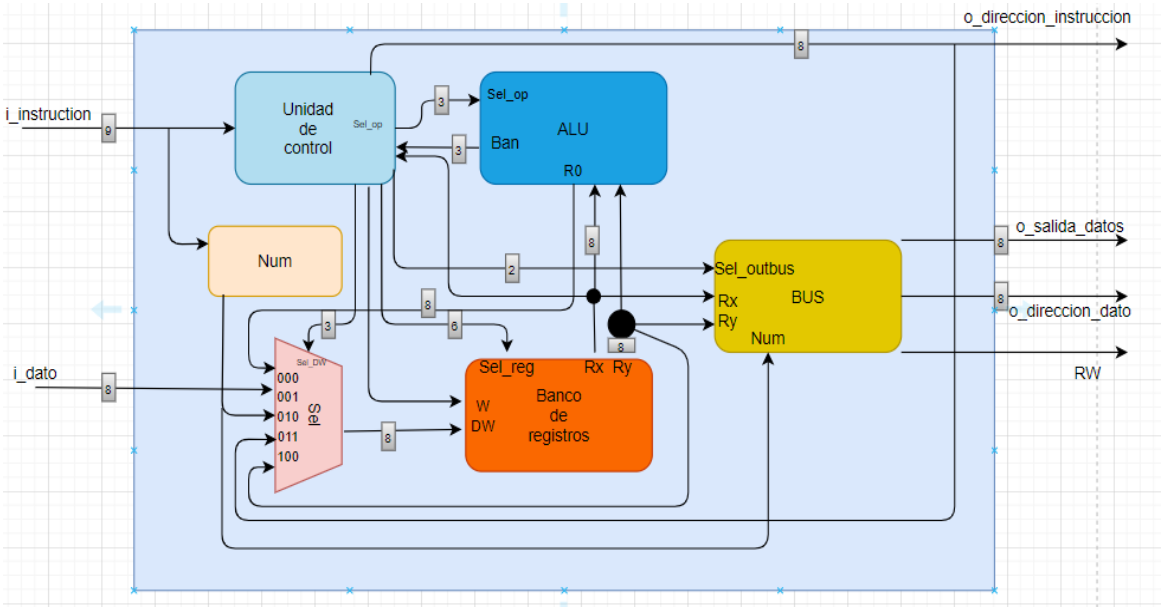


Instruction Set

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	

Arquitectura

Caja blanca microcontrolador

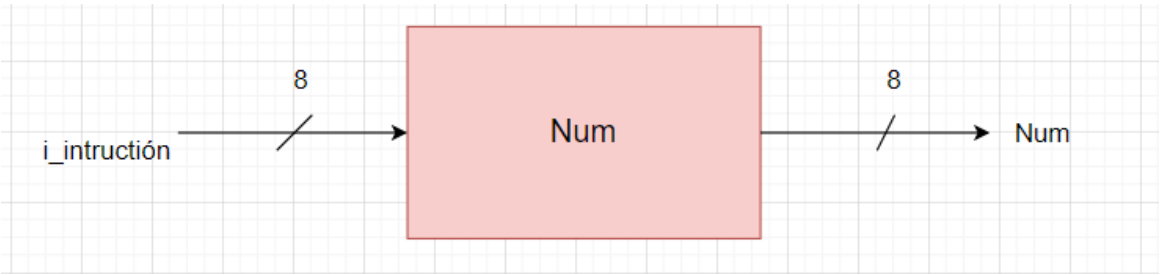


Señal	Dirección	Tamaño	Descripción
i_instrucción	Entrada	9	Señal de entrada la cual representa la instrucción codificada con el OP code y sus operandos, dependiendo de la instrucción nueva.
i_dato	Entrada	8	Entrada de datos proveniente de fuera del microprocesador.
o_dirección_instrucción	Salida	8	Señal de salida que representa la dirección de la instrucción que se va a ejecutar.
o_salida_datos	Salida	8	Dato que modifica la dirección de memoria.

o_dirección_dato	Salida	8	Indica a la dirección necesaria para la realización de una instrucción.
RW	salida	1	Indica si se lee o guarda en la memoria.

Num

Caja negra



Señal	Direccion	Tamaño	Descripción
i_instruction	Entrada	8	Señal de entrada que contiene la instrucción codificada
Num	Salida	8	Valor del dato directo correspondiente a un número para las instrucciones que requieran un valor directo.

Descripción funcional

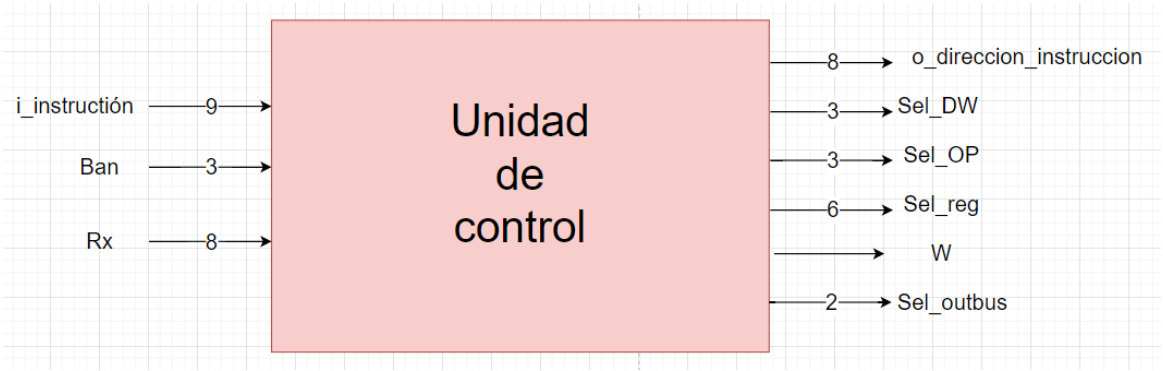
Modulo el cual toma los 3 bits menos significativos de la entrada "i_instruccion" para obtener un número dado por el valor del dato directo. (Especificar qué pasa con los otros 5 bits).

Tabla de verdad

Entrada	Salida
i_instrucction	i_instruccion=num i_instruccion = (0:2) 3 bits menos significativos num=(3:7) los bits restantes los pasa como ceros.

Unidad de control

Caja negra



Señal	Dirección	Tamaño	Descripción
i_instrución	Entrdada	9	Señal de entrada la cual representa la instrucción codificada con el OP code y sus operandos, dependiendo de la instrucción nueva
Ban	Entrada	3	Representa la bandera que generó la ALU
Rx	Entrada	8	Señal de entrada con un dato que proviene de un registro, que contiene una dirección para modificar el PC
o_direccion_instruccion	Salida	8	Señal de salida que representa la dirección de la instrucción que se va a ejecutar.
Sel_DW	Salida	3	Señal que selecciona qué dato se va a guardar en el banco de registros
Sel_OP	Salida	3	Señal que selecciona la operación que realizará la ALU
Sel_reg	Salida	6	Señal que selecciona los registros Rx y Ry en el banco de registros según la instrucción dada.
W	Salida	1	Señal que permite leer o escribir los datos provenientes de la unidad de control.
Sel_outbus	Salida	2	Señal que selecciona qué operación realizará el control de bus de salida

Descripción funcional

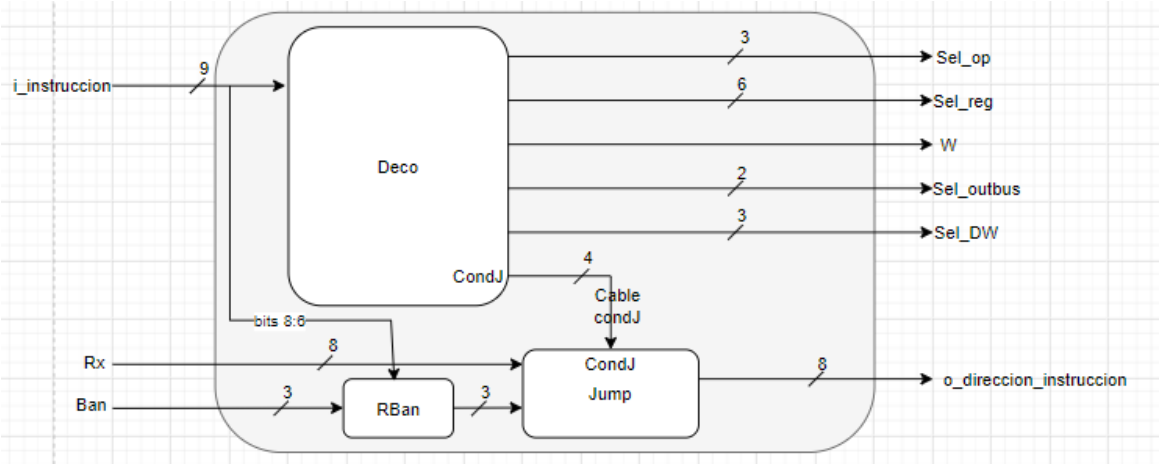
El módulo de control es el encargado de direccionar y decodificar las señales recibidas por parte del módulo num que previamente canalizo los bits de la señal de entrada "i_instruccion", para posteriormente enviar el dato correcto seleccionado a cada módulo para que ejecute respectivamente su tarea asignada.

Tabla de verdad

	i_instruccion	Sel_Op	Sel_reg	W	Sel_outbus	Sel_Dw	o_direccion_instruccion
Load	001, Rx, Num	0	000, Rx	1	00	010	Pc + 1
Load	010, Rx, Ry	0	Rx, Ry	1	01	001	Pc + 1
Store	011, Rx, Num	0	000, Rx	0	10	000	Pc + 1
Store	100, Rx, Ry	0	Ry, Rx	0	11	000	Pc + 1
Move	101, Rx, Ry	0	Ry, Rx	1	00	100	Pc + 1
Math	110, Rx, Op	Op	R0, Rx	1	00	000	Pc + 1
Jump	111, Rx, Cond	0	If Cond=1, a) 000,R7; 000, Rx; else 000, Rx	if Cond= 1; a) 1; b) 0; else 0	00	if Cond= 1; 011; else 000	Cond=0; Rx Cond=1; Pc = RX; Cond=2; if Z = 1; >>Rx; else >> Pc = R7 Cond=3; if Z = 0; >>Rx; else >> Pc Cond=4; if C= 1; >>Rx; else >> Pc Cond=5; if C = 0; >>Rx; else >> Pc Cond=6; if N= 1; >>Rx; else >> Pc Cond=7; if N = 0; >>Rx; else >> Pc
Nop	0	0	0	0	00	000	Pc

Deco

Caja negra



Descripción funcional

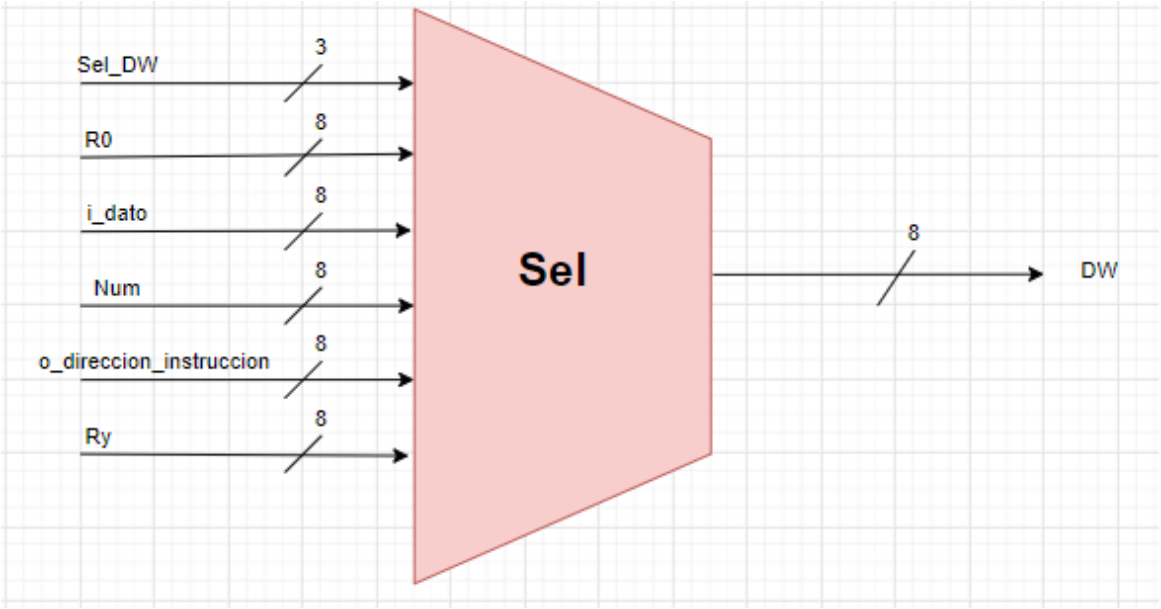
	i_instrucción [8:6], [5:3], [2:0]	Sel_Op	Sel_reg [5:3], [2:0]	Rw	Sel_output	Sel_Dw	CondJ
Load	001, Rx, Num	0	000, Rx	1	00	010	0001
Load	010, Rx, Num	0	Ry, Rx	1	01	001	0001
Store	011, Rx, Num	0	000, Rx	0	10	000	0001
Store	100, Rx, Ry	0	Ry, Rx	0	11	000	0001
Move	110, Rx, Ry	0	Rx, Ry	1	00	100	0001
Math	110, Rx, Op	Op	R0, Rx	1	00	000	0001
Jump	111, Rx, Cond	0	If Cond=1 a) 000, R7 b) 000,Rx else -> 0	If Cond =1 a) 1 b) 0 else ->0	00	If Cond=1 -> 011 else -> 000	{1'b1, Cond}
Nop	0	0	0	0	00	000	0001

Jump

Cond	CondJ	o_direccion_Instruc_Datos
000	1000	Rx
001	1001	Rx
010	1010	If Z= 1 -> Rx else -> Pc +1
011	1011	If Z= 0 -> Rx else -> Pc +1
100	1100	If C= 1 -> Rx else -> Pc +1
101	1101	If C= 0 -> Rx else -> Pc +1
110	1110	If N= 1 -> Rx else -> Pc +1
111	1111	If N= 0 -> Rx else -> Pc +1
No Jump	0001	Pc +1

Selector

Caja negra



Señal	Dirección	Tamaño	Decripción
Sel_DW	Entrada	3	Entrada del selector la cual nos indicará cual de los datos se guardará en el registro.
R0	Entrada	8	Es el resultado de la operación proveniente del modulo ALU, será guardado en el registro 0.
i_dato	Entrada	8	Entrada de datos proveniente de fuera del microprocesador.
Num	Entrada	8	Entrada la cual consiste en el valor del dato directo correspondiente a un numero para las instrucciones que requieran un valor directo.
o_dirección_instrucción	Entrada	8	Entrada de datos lque será guardada en el registro para poder tener un control de la instruccion en la que está.
Ry	Entrada	8	Entrada proveniente del registro Ry y volverá a ser guardada en el registro.
Dw	Salida	8	Salida del selector la cual se dirige hacia el banco de registros para ser guardada.

Descripción funcional

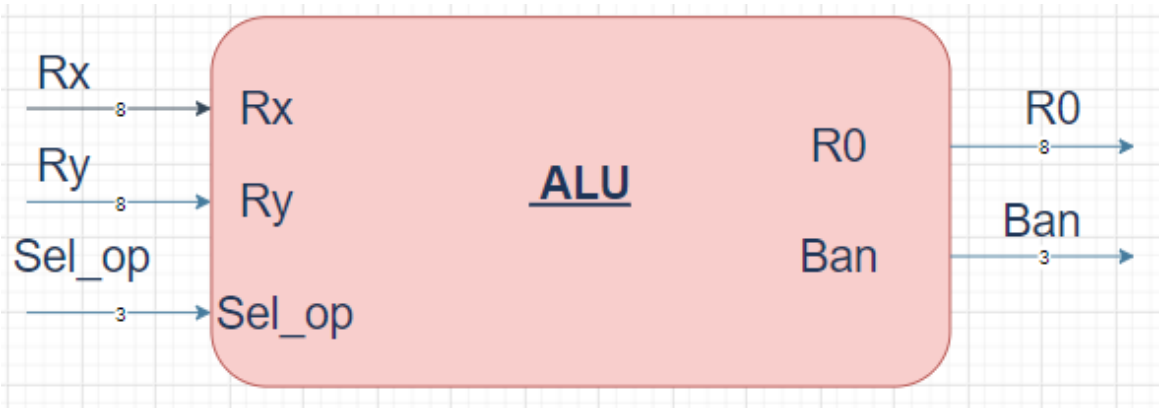
Módulo que contiene 8 registros los cuales almacenan la información necesaria misma que será utilizada posteriormente dependiendo de la indicación del controlador.

Tabla de verdad

Sel_Dw	Dw
000	R0
001	i_Dato
010	Num
011	o_direccion_intruccion
100	Ry
others	0

Alu

Caja negra



Señal	Dirección	Tamaño	Descripcion
Rx	Entrada	8	uno de los dos datos de 8 bits con el cual se llevaran las operaciones seleccionadas
Ry	Entrada	8	uno de los dos datos de 8 bits con el cual se llevaran las operaciones seleccionadas
Sel_op	Entrada	3	representa la selección de las operaciones a realizar las daso las dasos son OP: 0-Suma, 1-Resta, 2-Corrimiento a la izquierda, 3-Corrimiento a la derecha, 4-Negación mordida un poco, 5oY poco, 6o O un poco, 7xor poco un poco
R0	Salida	8	Dato de salida el cual representa el resultado de la operación llevada a cabo
Ban	Salida	3	Dato de salida que representa las banderas activadas por la salida Result

Descripción funcional

Módulo el cual realiza las operaciones lógicas con las entradas de los registros teniendo como salida el resultado de las operaciones y la activación de las banderas si el resultado es 0, hay un acarreo o es negativo.

Tabla de verdad

in_Sel_Op	Combinacion Bin Sel_Op	Operación designada
0	000	R0+Rx
1	001	R0-Rx
2	010	Desplazar a la izq Rx binario de R0
3	011	Desplazar a la izq Ry binario de R0
4	100	R0= binario neg de Rx
5	101	AND (R0 y Rx)
6	110	Or (R0 y Rx)
7	111	Xor (R0 y Rx)

Banderas

Ban	valor	Interpretación
Z	0	Si el resultado no es 0
	1	si el resultado es 0
C	0	si el resultado no es negativo
	1	si el resultado es negativo
N	0	si no lleva un carry
	1	si lleva el carry

Banco de registros

Caja negra

Señal	Dirección	Tamaño	Descripción
Sel_reg	Entrada	6	Es la entrada que permite seleccionar el registro donde se realizará la instrucción.
W	Entrada	1	Entrada que permite leer o escribir los datos provenientes de la unidad de control.
DW	Entrada	8	Entrada con el dato proveniente del selector que viene del banco de registros, el cual se almacenará en alguno de los 8 registros.
Rx	Salida	8	Dato de salida de 8 bits, el cual representa uno de los dos datos existentes, con los cuales se realizará la operación seleccionada.
Ry	Salida	8	Dato de salida de 8 bits, el cual representa uno de los dos datos existentes, con los cuales se realizará la operación seleccionada.

Descripción funcional

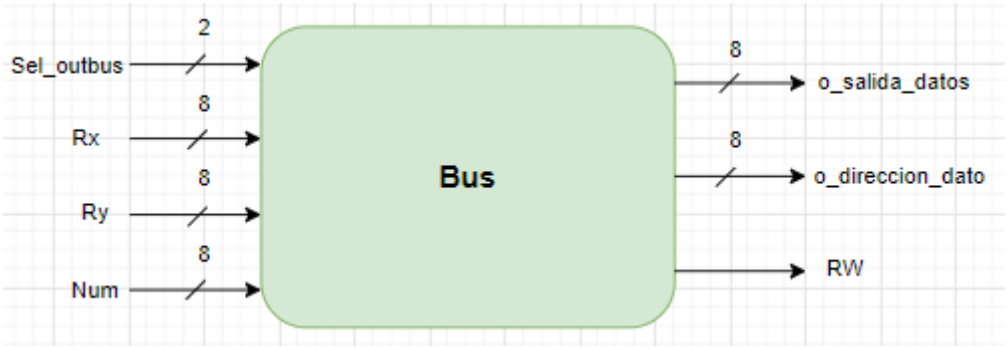
El banco de registros almacena datos de pocos bits para utilizarlos después dependiendo lo que se necesite según el módulo de control.

Tabla de verdad

W	Rx	Ry
1	Sel_reg(2:1) igual a DW	Sel_reg(5:3)
0	Sel_reg(2:1)	Sel_reg(5:3)

Bus

Caja negra



Senal	Direccion	Tamano de bits	Descripcion
Sel_outbus	Entrada	2	Encargada de seleccionar la operacion a realizar en la salida
Rx	Entrada	8	selecciona un dato de los 2 con lo cuales se realizara la operación establecida.
Ry	Entrada	8	selecciona un dato de los 2 con lo cuales se realizara la operación establecida.
Num	Entrada	8	Dato de entrada para instruccion store de numero inmediato
o_salida_datos	Salida	8	Dato que modifica la dirección de memoria
o_direccion_dato	Salida	8	Indica al direccion necesaria para la realizacion de una instruccion
Rw	Salida	1	Indica si se lee o guarda en la memoria

Descripción funcional

Manipula todas las acciones del bus ya sea la dirección, información y si va a leer o escribir.

Tabla de verdad

Instruccion	i_Sel_outbus	o_salida_datos	o_direccion_dato	Rw
Nop	00	0	0	0
Load Ry	01	0	Rx	0
Store, Num, Rx	10	Num	Rx	1
Store, Ry, Rx	11	Ry	Rx	1

Implementación

Unidad de cotrol

El modulo de control es el encargado de direccionar y decodificar las señales recibidas por parte del modulo num que previamente canalizo los bits de la señal de entrada "i_instruccion", para posteriormente enviar el dato correcto seleccionado a cada modulo para que ejecute respectivamente su tarea asignada.

Alu

Módulo el cual realiza las operaciones lógicas con las entradas de los registros teniendo como salida el resultado de las operaciones y la activación de las banderas si el resultado es 0, hay un acarreo o es negativo.

Banco de registros

El banco de registros almacena datos de pocos bits parabospues dependiendo lo que se cree según el modulo de control.

Selector

Módulo que conteo 8 registros los registros la información la información misma que será utilizada posterior dependiendo de la indicación del controlador.

Bus

manipular todas las acciones del bus ya sea la dirección, informacion y si va a leer o escribir

Memoria Ram

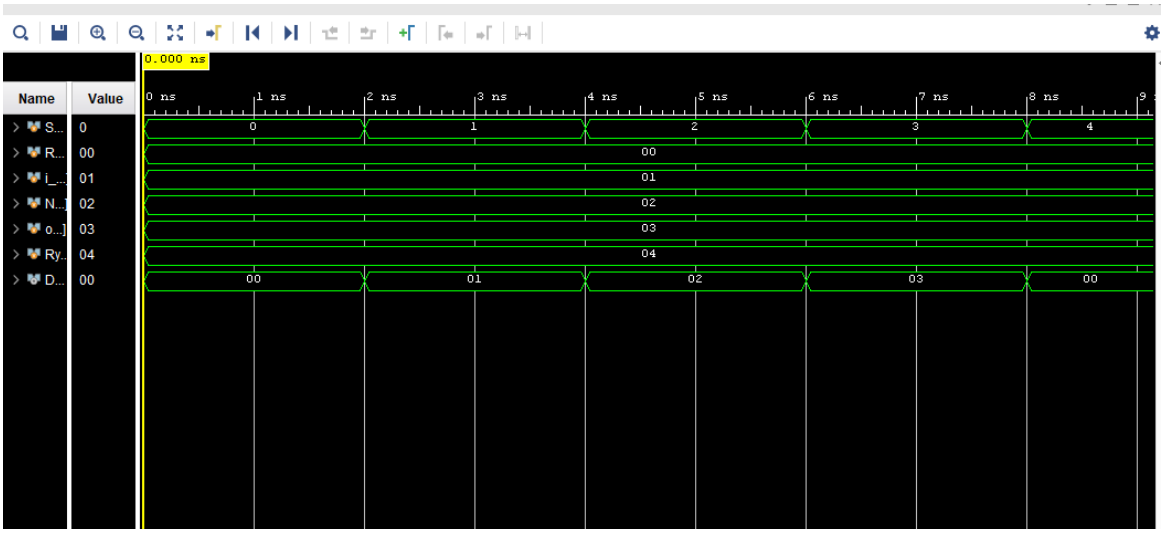
Para los datos que se requieran en el microcontrolador.

Rom

Ingresan intrucciones requeridas por el microcontrolador.

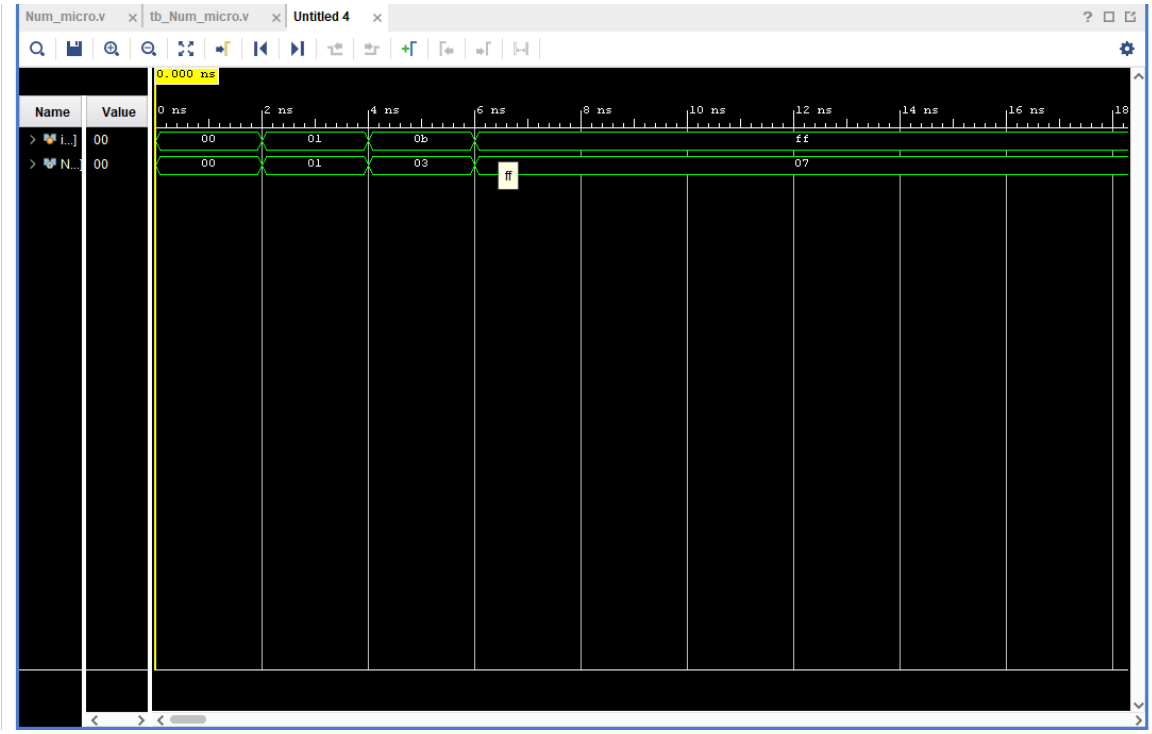
Pruebas

Selector

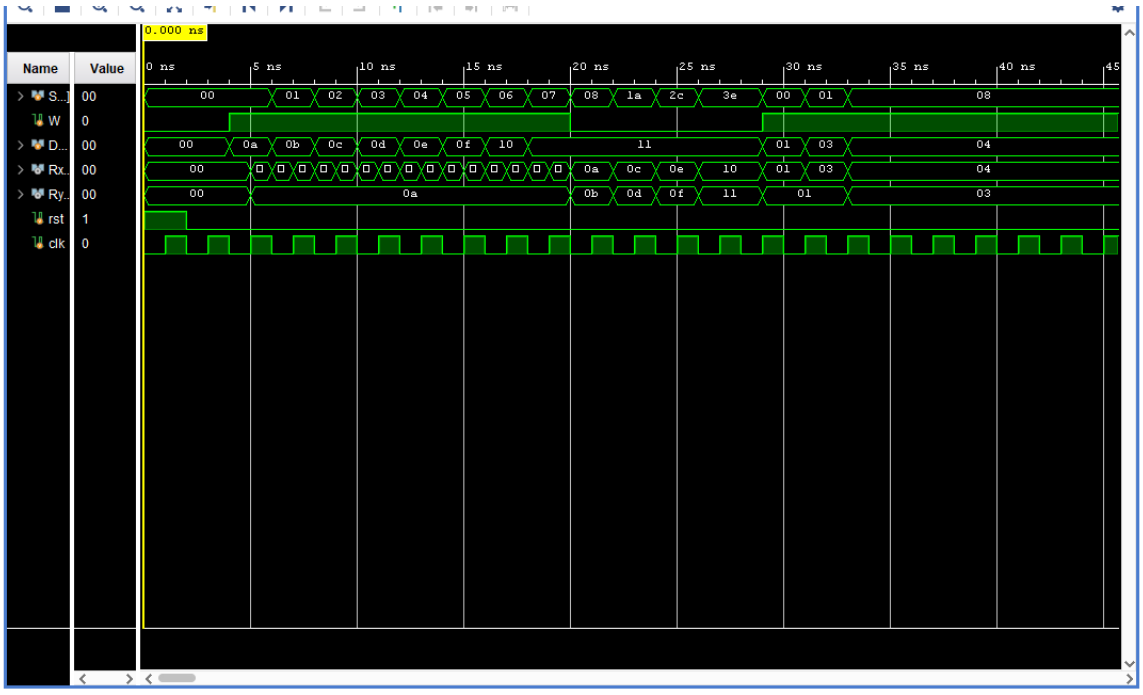


Bus

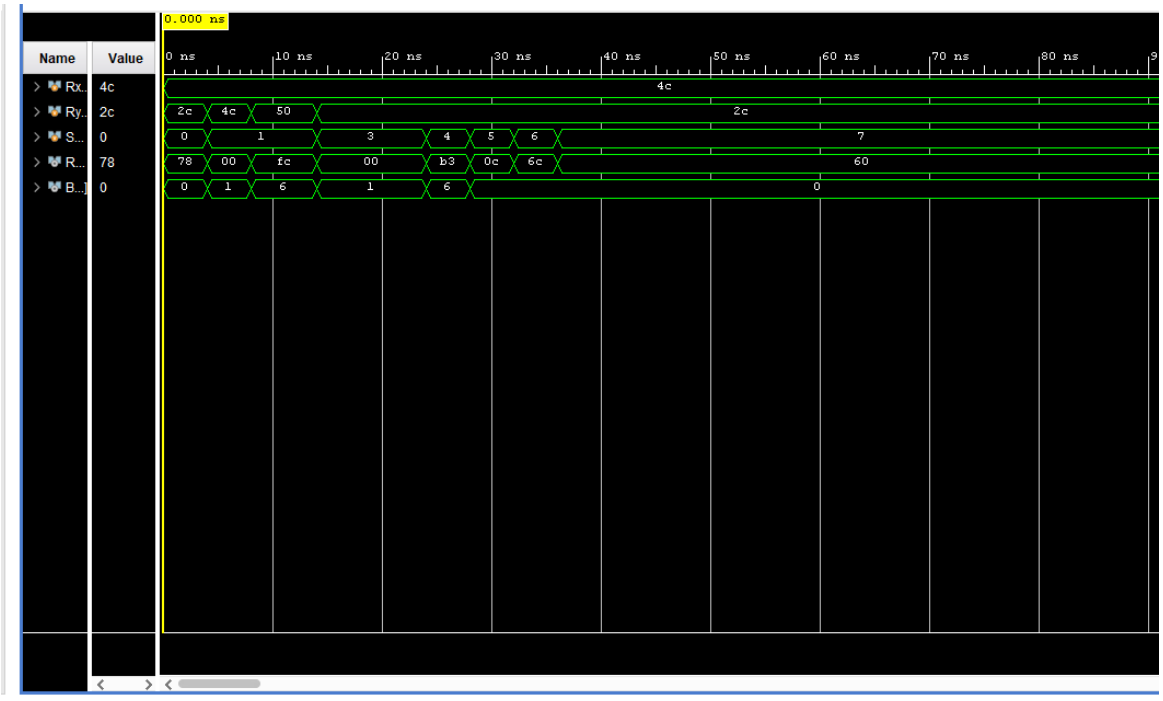
Num


Num_micro.v
tb_Num_micro.v
Untitled 4
0.000 ns
Name
Value
0 ns
2 ns
4 ns
6 ns
8 ns
10 ns
12 ns
14 ns
16 ns
18
00
00
00
01
0b
ff
00
01
03
07
ff

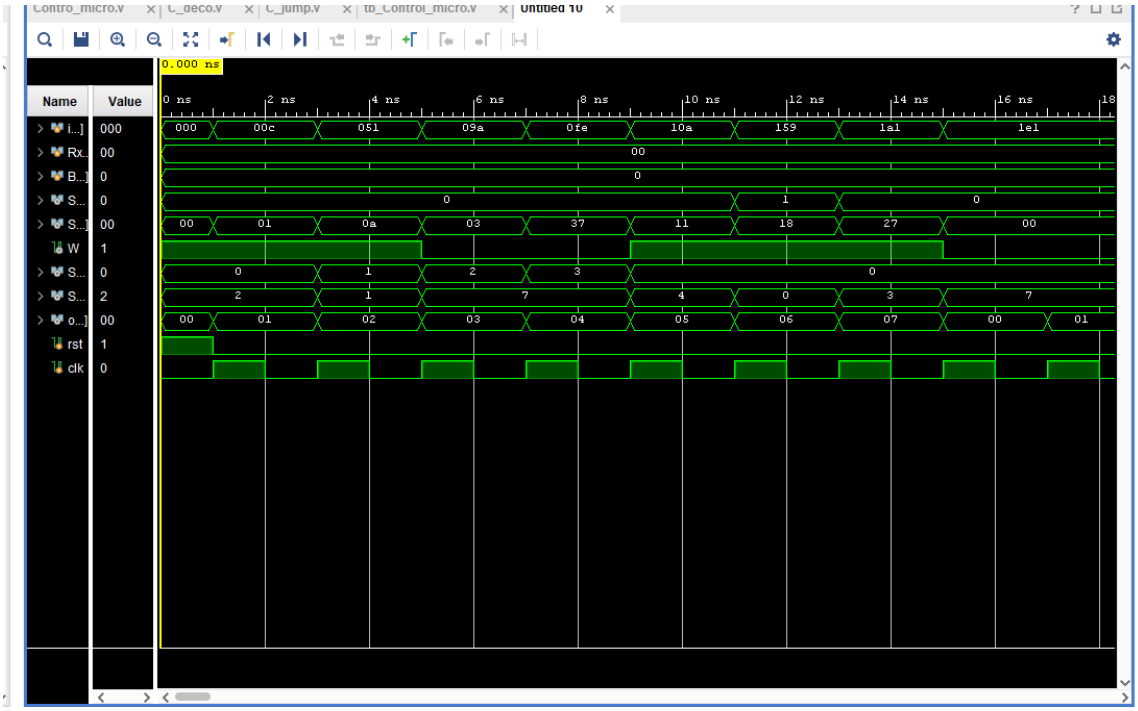
Banco de registros



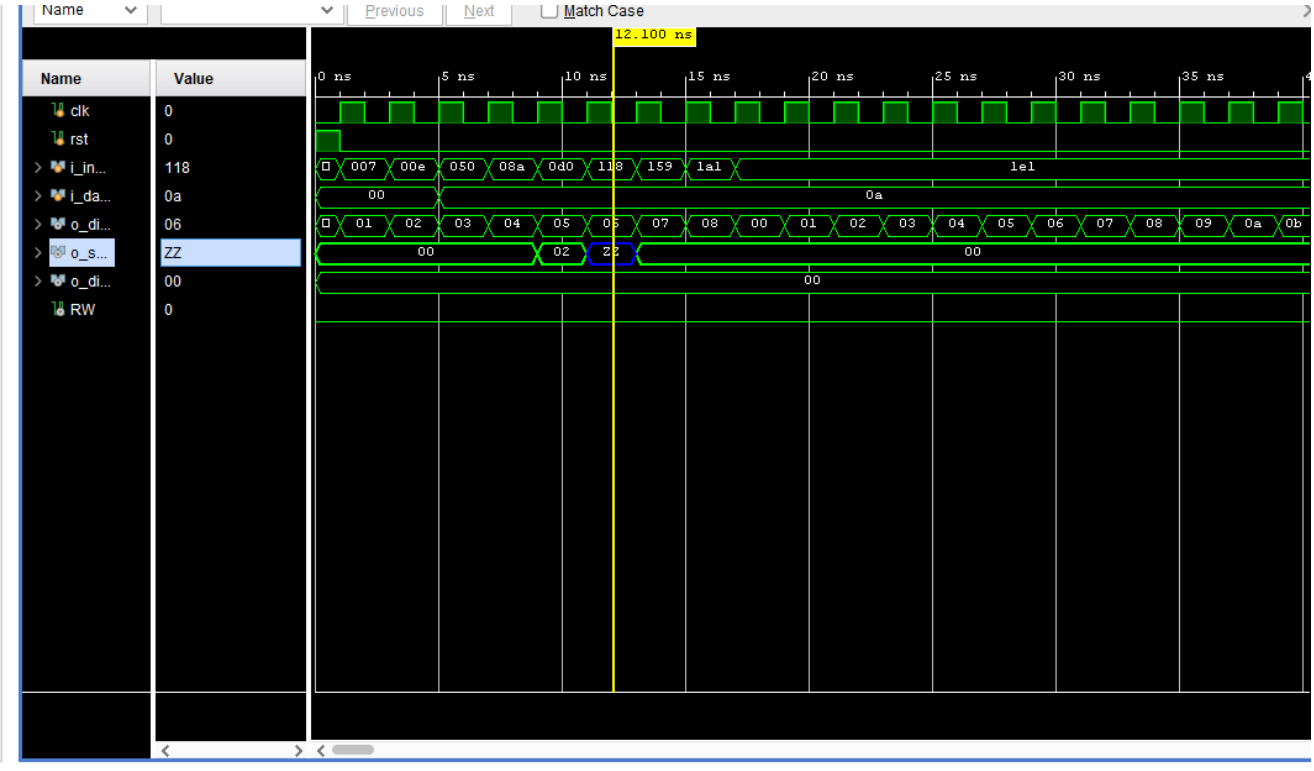
Alu



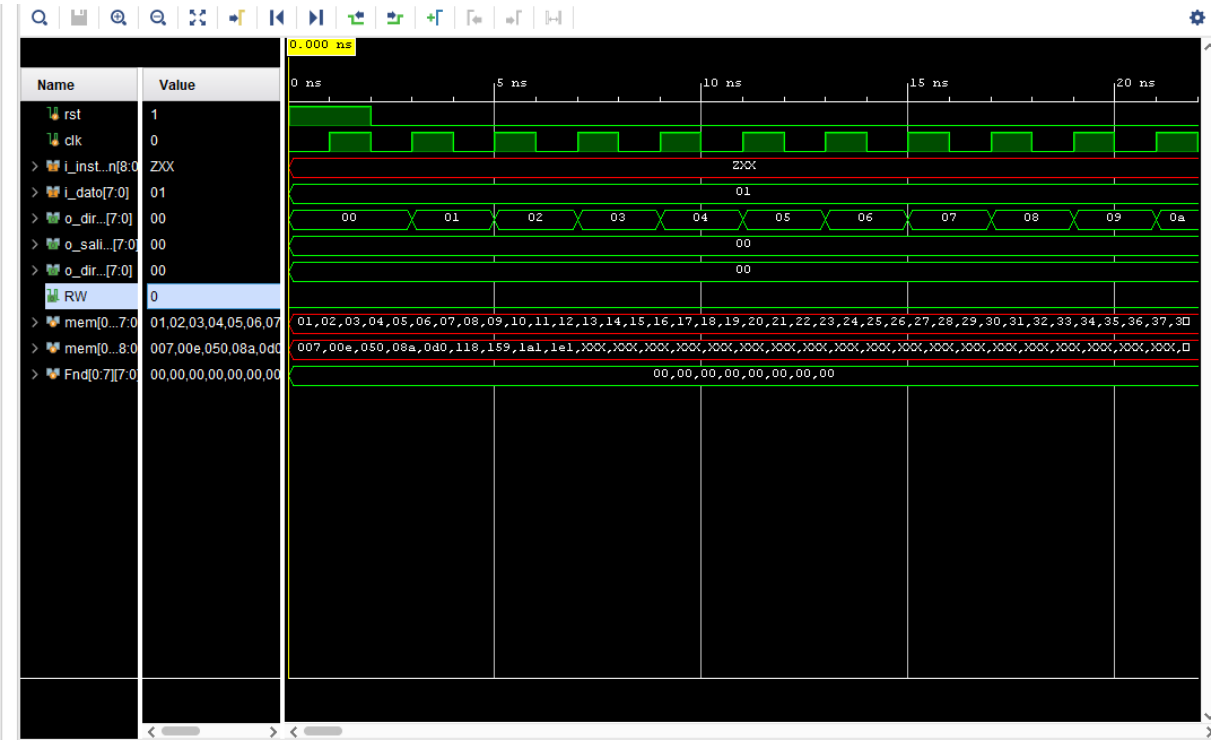
Jump



Micro8bits_Harvard



Compu_personal



Análisis de resultados

Ya habiendo finalizado el sistema, con buen funcionamiento el microcontrolador, una memoria de datos y de instrucciones, diseñamos programas con a base de las instrucciones establecidas se realizaron una multiplicación y división, almacenado el dato obtenido en un registro de la memoria de datos para probar el que este funcionando de manera apropiada o ver si algo anda fallando.

Diagrama de División y multiplicación:

Tabla de multiplicacion

Direccion de intruccion	Instruccion	Octal
0	Load R1 Num	01X
1	Load R2 Num	02X
2	Load R3 1	031
3	Load R4 7	047
4	Load R5 [5]	155
5	Math R2 0	520
6	Jump R5 2	652
7	Move R0 R6	406
8	Math R1 0	510
9	Move R6 R0	460
10	Move R0 R2	402
11	Math R3 1	531
12	Move R2 R0	420
13	Jump R4 3	643
14	Move R0 R6	406
15	No operation	7XX

Tabla de division

Direccion de intruccion	Instruccion	Octal
0	Load R1 Num	01X
1	Load R2 Num	02X
2	Load R3 1	031
3	Load R4 4	044
4	Move R0 R5	405
5	Math R3 0	530
6	Move R5 R0	450
7	Move R0 R1	401
8	Math R2 1	521
9	Move R1 R0	410
10	Jump R4 7 if N=0	647
11	Move R0 R5	405
12	Math R3 1	531
13	No operation	7XX

Diagrama de Multiplicación

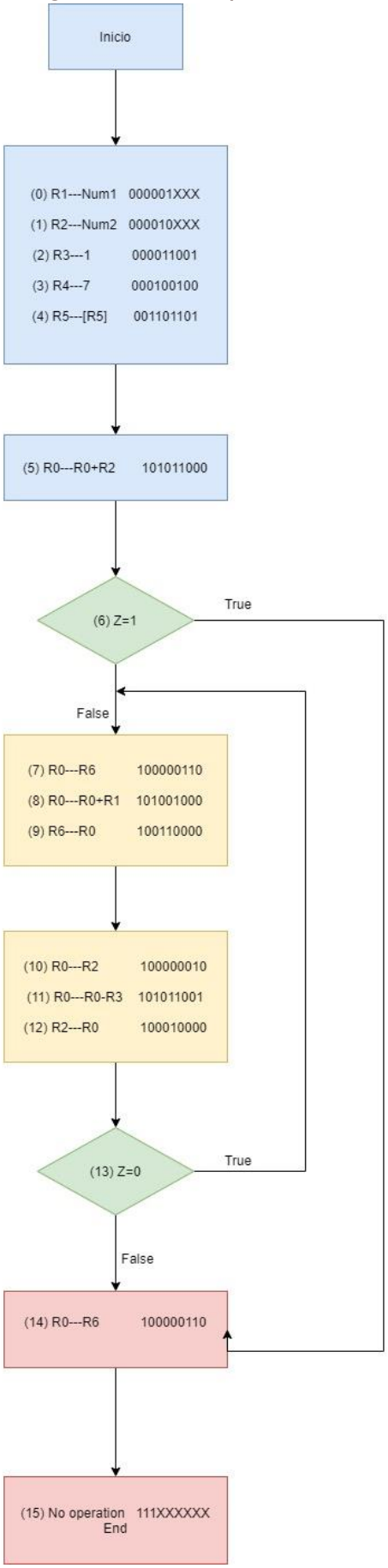
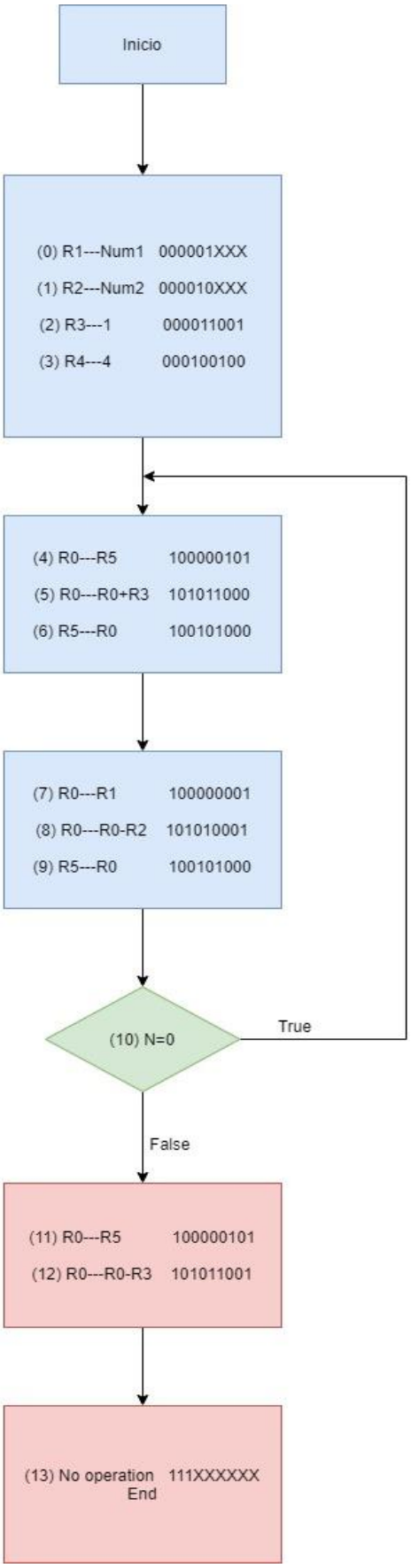


Diagrama de Division



Conclusiones

Gracia a los datos obtenidos podemos ver que para hacer un microcontrolador se requiere aplicar los conocimientos adquiridos a través del transcurso de sistemas digitales 1 y 2, mas no obstante aun con esos conocimientos no será suficiente para lograrlo. Conociendo mejor un microcontrolador sabremos que cuenta con datos que entran y salen, pero aún hay más si profundizamos veremos que internamente está compuesto por una unidad de control la cual se encarga de recibir instrucciones decodificadas y efectúa coordinadamente las acciones de los otros submódulos internos, un Núm. que toma los 3 bits mas significativos de una instrucción para la obtención del dato correspondiente a un número, un banco de registros que almacena información para despues utilizarla en el selector que seleccionara un dato guardado o almacenado por la instrucción que recibe del controlador, también constara de un ALU (Unidad Aritmética Lógica) que realizara operaciones mandadas de la unidad de control del microprocesador, un bus se encargar de las direcciones e información de los datos de salida. Gracias a esto desarrollaremos un microcontrolador.

Referencias

[1] M. Morris Mano, *Diseño Digital 3ra Edición*. California State University, Los Ángeles. Pearson Educación, 2003.

[2] Thomas L. Floyd, Fundamentos de Sistemas Digitales 9na Edición. Madrid. Pearson Education, 2006

[3] H. Zea, J. David y Alfredo. "Design and implementation of a special-purpose microprocessor", 2011.

[4] GitHub - (23 de septiembre del 2020) [Internet Archivo PDF] Disponible en: https://github.com/remberto-uaz/Rob7MicrosAgo2020/blob/master/MicroUAZ_Ago2020/microUAZ8-2020v0_02.pdf [Acceso el 15 de Octubre del 2020].

Código de implementación

Compu_personal

```
module proyecto_pc
    (
     input rst,
     input clk
    );

    wire [8:0] i_instruccion;
    wire [7:0] i_dato;
    wire [7:0] o_direccion_instruccion;
    wire [7:0] o_salida_datos;
    wire [7:0] o_direccion_datos;
    wire RW;

    Micro8bits_Harvard compu (
        .clk(clk),
        .rst(rst),
        .i_instruccion(i_instruccion),
        .i_dato(i_dato),
        .o_direccion_instruccion(o_direccion_instruccion),
        .o_salida_datos(o_salida_datos),
        .o_direccion_datos(o_direccion_datos),
        .RW(RW)
        );

        M_Ram #(8,8,256) memDts (
```

```
            .e(o_salida_datos),
            .a(o_direccion_datos),
            .RW(RW),
            .s(i_dato)
            );

        M_Rom #(9,8,256) memInstr (
            .a(o_direccion_instruccion),
            .s(i_instruccion)
            );
endmodule
```

Micro8bits_Harvard

```
module Micro8bits_Harvard
(
    input clk,
    input rst,
    input [8:0] i_instruccion,
    input [7:0] i_dato,
    output [7:0] o_direccion_instruccion,
    output [7:0] o_salida_datos,
    output [7:0] o_direccion_datos,
    output RW
    );

    /////Conexiones internas del microposesador 8 bits harvard////
    wire [7:0] c_Num;
    wire [2:0] c_DW;
    wire [5:0] c_Sel_reg;
```

```
wire c_W;
wire [7:0] c_o_DW;
wire [7:0] c_Rx;
wire [7:0] c_Ry;
wire [2:0] c_Sel_op;
wire [7:0] c_R0;
wire [2:0] c_Ban;
wire [1:0] c_Sel_outbus;

////////// modulo de Control_micro.v /////////
Control_micro
Control(
      .i_instruccion(i_instruccion),
      .Rx(c_Ry),
      .Ban(c_Ban),
      .Sel_op(c_Sel_op),
      .Sel_reg(c_Sel_reg),
      .W(c_W),
      .Sel_outbus(c_Sel_outbus),
      .Sel_DW(c_o_DW),
      .o_direccion_instruccion(o_direccion_instruccion),
      .rst(rst),
      .clk(clk)
    );
////////// modulo de alu_micro ////////////
alu_micro
Alu (
    .Rx(c_Rx),
```

```
   .Ry(c_Ry),
   .Sel_op(c_Sel_op),
   .R0(c_R0),
   .Ban(c_Ban)
 );
////////// modulo de Num_mirco //////////
Num_micro
    numero (
        .i_instruccion(i_instruccion),
        .Num(c_Num)
        );
////////// modulo de selector_micro ////////
selector_micro
selector (
    .Sel_DW(c_o_DW),
     .R0(c_R0),
     .i_dato(i_dato),
     .Num(c_Num),
     .o_direccion_instruccion(o_direccion_instruccion),
     .Ry(c_Ry),
     .DW(c_DW)
     );
////////// modulo de ban_reg_micro /////////
ban_reg_micro
    banco_registros (
        .Sel_reg(c_Sel_reg),
        .W(c_W),
        .DW(c_DW),
```

```
            .Rx(c_Rx),
            .Ry(c_Ry),
            .rst(rst),
            .clk(clk)
            );
    ////////// modulo de Bus_micro ////////////
    Bus_micro
        Bus(
         .Sel_outbus(c_Sel_outbus),
         .Rx(c_Rx),
         .Ry(c_Ry),
         .Num(c_Num),
         .o_salida_datos(o_salida_datos),
         .o_direccion_datos(o_direccion_datos),
         .RW(RW),
         .rst(rst),
         .clk(clk)
         );


endmodule
```

Selector

```
module selector_micro
(
    input [2:0] Sel_DW,
    input [7:0] R0,
    input [7:0] i_dato,
    input [7:0] Num,
```

```
    input [7:0] o_direccion_instruccion,
    input [7:0] Ry,
    output reg [7:0] DW
    );

    always@*
        begin
            case(Sel_DW)
                3'b000 : begin DW <= R0; end
                3'b001 : begin DW <= i_dato; end
                3'b010 : begin DW <= Num; end
                3'b011 : begin DW <= o_direccion_instruccion; end
                3'b011 : begin DW <= Ry; end
                default : begin DW <= 0; end
            endcase
        end
endmodule
```

Bus

```
module Bus_micro
    (
    input [1:0] Sel_outbus,
    input [7:0] Rx,
    input [7:0] Ry,
    input [7:0] Num,
    output reg [7:0] o_salida_datos,
```

```
output reg [7:0] o_direccion_datos,
output reg RW,
input rst,
input clk
);

always@(posedge clk, posedge rst)
    begin
        if(rst)
            begin
                o_salida_datos <= 0;
                o_direccion_datos <= 0;
                RW <= 0;
            end
        else
            case(Sel_outbus)
                2'b00: begin o_salida_datos <= 0; end
                2'b00: begin o_direccion_datos <= 0; end
                2'b00: begin RW <=0;
                end
                2'b01: begin o_salida_datos <= 0; end
                2'b01: begin o_direccion_datos <= Ry; end
                2'b01: begin RW <=0;
                end
                2'b10: begin o_salida_datos <= Num; end
                2'b10: begin o_direccion_datos <= Rx; end
                2'b10: begin RW <=1;
                end
```

```
                2'b11: begin o_salida_datos <= Ry; end
                2'b11: begin o_direccion_datos <= Rx; end
                2'b11: begin RW <= 1;
                end
                default:
                   begin
                      o_salida_datos <= 0;
                      o_direccion_datos <= 0;
                      RW <= 0;
                   end
             endcase
       end
endmodule
```

Num

```
module Num_micro
#(parameter DW = 8)
   (
   input [(DW -1) : 0] i_instruccion,
   output reg [(DW -1) : 0] Num
   );

   reg [(DW -1) : 0] NM;

      always@*
         begin
         NM = 8'b0;
```

```
            Num <= {NM[7:3], i_instruccion[2:0]};
        end
endmodule
```

Banco de registros

```
module ban_reg_micro
(
    input [5:0] Sel_reg,
    input W,
    input [7:0] DW,
    output [7:0] Rx,
    output [7:0] Ry,
    input rst,
    input clk
    );

    reg [7:0] Fnd [0:7];

    always@(posedge clk, posedge rst)
     begin
        if(rst)
            begin
                Fnd[0]<=0;
                Fnd[1]<=0;
                Fnd[2]<=0;
                Fnd[3]<=0;
                Fnd[4]<=0;
```

```
            Fnd[5]<=0;
            Fnd[6]<=0;
            Fnd[7]<=0;
        end
    else
        if (W)
          Fnd[Sel_reg[2:0]]<=DW;
        end
        assign Rx= Fnd[Sel_reg[2:0]];
        assign Ry= Fnd[Sel_reg[5:3]];
endmodule
```

Alu

```
module alu_micro
(
   input [7:0] Rx,
   input [7:0] Ry,
   input [2:0] Sel_op,
   output [7:0] R0,
   output [2:0] Ban
   );

   reg [8:0] F;
```

```
        always@*
            begin
                case(Sel_op)
                    0: F <= Rx+Ry;
                    1: F <= Rx-Ry;
                    2: F <= Rx<<Ry;
                    3: F <= Rx>>Ry;
                    4: F <= ~Rx;
                    5: F <= Rx&Ry;
                    6: F <= Rx|Ry;
                    7: F <= Rx^Ry;
                endcase
            end
    assign R0 = F[7:0];
    assign Ban [0] = &(~F);
    assign Ban [1] = F[8];
    assign Ban [2] = F[7];
endmodule
```

Control

```
module Control_micro
(
    input [8:0] i_instruccion,
    input [7:0] Rx,
    input [2:0] Ban,
    output [2:0] Sel_op,
    output [5:0] Sel_reg,
```

```
output W,
output [1:0] Sel_outbus,
output [2:0] Sel_DW,
output [7:0] o_direccion_instruccion,
input rst,
input clk
);

wire [3:0] cable_condJ;
reg [2:0] RBan;

deco C_deco
(
    .i_instruccion(i_instruccion),
    .condJ(cable_condJ),
    .Sel_op(Sel_op),
    .Sel_reg(Sel_reg),
    .W(W),
    .Sel_outbus(Sel_outbus),
    .Sel_DW(Sel_DW),
    .rst(rst),
    .clk(clk)
    );

jump C_jump
(
    .condJ(cable_condJ),
    .Rx(Rx),
```

```
        .Ban(RBan),
        .o_direccion_instruccion(o_direccion_instruccion),
        .rst(rst),
        .clk(clk)
        );

    always @(posedge clk, posedge rst)
        begin
            if(rst)
                begin
                    RBan<=0;
            end
        else
            if (i_instruccion[8:6]==3'b101)
                RBan<=Ban;
    end

endmodule
```

Deco

```
module deco(
    input [8:0] i_instruccion,
    output reg [3:0] condJ,
    output reg [2:0] Sel_op,
    output reg [5:0] Sel_reg,
    output reg W,
    output reg [1:0] Sel_outbus,
```

```
    output reg [2:0] Sel_DW,
    input rst,
    input clk
    );


    parameter
    LoadRxNum =3'b000,
    LoadRxARy =3'b001,
    StoreARxNum =3'b010,
    StoreARxRy =3'b011,
    MoveRxRy =3'b100,
    MathRxOp =3'b101,
    JumpRxCond =3'b110,
    NOP =3'b111;


    always @(i_instruccion)
        begin
        case (i_instruccion[8:6])
        LoadRxNum: begin   condJ <= 4'b0001; Sel_op <= 0; Sel_reg <=
{3'b000,i_instruccion[5:3]}; W <= 1; Sel_outbus <= 0; Sel_DW <= 3'b010; end
        LoadRxARy: begin   condJ <= 4'b0001; Sel_op <= 0; Sel_reg <=
{i_instruccion[2:0],i_instruccion[5:3]}; W <= 1; Sel_outbus <= 2'b01;
Sel_DW<=3'b001; end
        StoreARxNum: begin condJ <= 4'b0001; Sel_op <= 0; Sel_reg <=
{3'b000,i_instruccion[5:3]}; W <= 0; Sel_outbus <= 2'b10; Sel_DW <= 3'b111; end
        StoreARxRy: begin  condJ <= 4'b0001; Sel_op <= 0; Sel_reg <=
{i_instruccion[2:0],i_instruccion[5:3]}; W <= 0; Sel_outbus <= 2'b11; Sel_DW <=
3'b111; end
```

```
        MoveRxRy: begin    condJ <= 4'b0001; Sel_op <= 0; Sel_reg <= {i_instruccion[2:0],i_instruccion[5:3]}; W <= 1; Sel_outbus <= 0; Sel_DW <= 3'b100; end

        MathRxOp: begin    condJ <= 4'b0001; Sel_op <= i_instruccion[2:0]; Sel_reg <= {i_instruccion[5:3],3'b000}; W <= 1; Sel_outbus <= 0; Sel_DW <= 3'b000; end

        JumpRxCond: begin

            if(i_instruccion[2:0]==3'b001) // jump sin condicion y guardar pc en R7

                begin  condJ <= {1'b1,i_instruccion[2:0]}; Sel_op <= 0; Sel_reg <= {i_instruccion[5:3],3'b111}; W <= 1; Sel_outbus <= 0; Sel_DW <= 3'b011;  end

            else //las demas condiciones de jump

                begin  condJ <= {1'b1,i_instruccion[2:0]}; Sel_op <= 0; Sel_reg <= {i_instruccion[5:3],3'b000}; W <= 0; Sel_outbus <= 0; Sel_DW <= 3'b111;  end

            end

        NOP:begin      condJ <= 4'b0001; Sel_op <= 0; Sel_reg <= 0; W <= 0; Sel_outbus <= 0; Sel_DW <= 3'b111; end

        default:begin  condJ <= 4'b0001; Sel_op <= 0; Sel_reg <= 0; W <= 0; Sel_outbus <= 0; Sel_DW <= 3'b111; end

        endcase

    end


endmodule
```

Jump

```
module jump

(

    input [3:0] condJ,

    input [7:0] Rx,

    input [2:0] Ban,

    output [7:0] o_direccion_instruccion,

    input rst,
```

```
input clk
);

reg [7:0] PC;

always @(posedge clk, posedge rst)
    begin
        if(rst)
            PC <= 0;
        else
          case (condJ)
            4'b0000:
                PC <= PC;
            4'b0001: // etapa de decodificacion de intruccion
                PC <= PC+1'b1;
            4'b1000:
                PC <= Rx;
            4'b1001:
                PC <= Rx;
            4'b1010:
                if(Ban[0])
                    PC <= Rx;
                else
                    PC <= PC+1'b1;
            4'b1011:
                if(~Ban[0])
                    PC <= Rx;
                else
```

```
                PC <= PC+1'b1;
            4'b1100:
                if(Ban[2])
                    PC <= Rx;
                else
                    PC <= PC+1'b1;
            4'b1101:
                if(~Ban[2])
                    PC <= Rx;
                else
                    PC <= PC+1'b1;
            4'b1110:
                if(Ban[1])
                    PC <= Rx;
                else
                    PC <= PC+1'b1;
            4'b1111:
                if(~Ban[1])
                    PC <= Rx;
                else
                    PC <= PC+1'b1;
            default: PC <= PC+1'b1;
        endcase
    end
    assign o_direccion_instruccion = PC;

endmodule
```

Test benches

Compu_personal

```
module tb_compu_personal;
    reg rst;
    reg clk;

    proyecto_pc uut (
     .rst(rst),
     .clk(clk)
    );

    initial
        begin
            rst = 1;
            clk = 0;

            #2 rst = 0;
            #20 $finish;
        end
    always
        #1 clk = !clk;
Endmodule
```

Micro8bits_Harvard

```
module tb_Micro8bits_Harvard;

    reg clk;
```

```
reg rst;
reg [8:0] i_instruccion;
reg [7:0] i_dato;
wire [7:0] o_direccion_instruccion;
wire [7:0] o_salida_datos;
wire [7:0] o_direccion_datos;
wire RW;


Micro8bits_Harvard uut
(
 .clk(clk),
 .rst(rst),
 .i_instruccion(i_instruccion),
 .i_dato(i_dato),
 .o_direccion_instruccion(o_direccion_instruccion),
 .o_salida_datos(o_salida_datos),
 .o_direccion_datos(o_direccion_datos),
 .RW(RW)
 );

initial
    begin
        rst=1;
        clk=0;
        i_instruccion = 0;
        i_dato = 0;
```

```
        #1 rst = 0; i_instruccion = 9'b000_000_111; //load Rx Num
        #2 i_instruccion = 9'b000_001_110; //load Rx Num
        #2 i_instruccion = 9'b001_010_000; // load Rx [Ry]
          i_dato = 10;
        #2 i_instruccion = 9'b010_001_010; // store [Rx] Num
        #2 i_instruccion = 9'b011_010_000; // store [Rx] Ry
        #2 i_instruccion = 9'b100_011_000; // move Rx Ry
        #2 i_instruccion = 9'b101_011_001; // Math Rx OP
        #2 i_instruccion = 9'b110_100_001; // Jump [Rx] Cond
        #2 i_instruccion = 9'b111_100_001; // Nop

    end

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

Selector

```
module tb_selector_micro;

  reg [2:0] Sel_DW;
  reg [7:0] R0;
  reg [7:0] i_dato;
  reg [7:0] Num;
  reg [7:0] o_direccion_instruccion;
  reg [7:0] Ry;
  wire [7:0] DW
  ;
```

```
  selector_micro uut
  (
       .Sel_DW(Sel_DW),
       .R0(R0),
       .i_dato(i_dato),
       .Num(Num),
       .o_direccion_instruccion(o_direccion_instruccion),
       .Ry(Ry),
       .DW(DW)
  );

initial
  begin
      Sel_DW = 3'd0;
      R0 = 8'd0;
      Num = 8'd2;
      i_dato= 8'd1;
      Ry =8'd4;
      o_direccion_instruccion = 8'd3;



      #2
        Sel_DW = 3'd1;
      #2
        Sel_DW = 3'd2;
      #2
        Sel_DW = 3'd3;
```

```
        #2
          Sel_DW = 3'd4;
    end
endmodule
```

Bus

```
module tb_Bus_micro;
    reg [1:0] Sel_outbus;
    reg [7:0] Rx;
    reg [7:0] Ry;
    reg [7:0] Num;
    wire [7:0] o_salida_datos;
    wire [7:0] o_direccion_datos;
    wire RW;
    reg rst;
    reg clk
    ;

    Bus_micro uut
      (
       .Sel_outbus(Sel_outbus),
       .Rx(Rx),
       .Ry(Ry),
       .Num(Num),
       .o_salida_datos(o_salida_datos),
       .o_direccion_datos(o_direccion_datos),
       .RW(RW),
```

```
        .rst(rst),
        .clk(clk)
        );

        initial
          begin
            Sel_outbus = 0;
            Rx = 0;
            Ry = 0;
            Num = 0;
            rst = 1;
            clk = 0;

            #2 rst = 0; Rx = 8'd5; Ry = 8'd6; Num = 8'd2; Sel_outbus = 0;
            #2 Sel_outbus = 2'b01;
            #2 Sel_outbus = 2'b10;
            #2 Sel_outbus = 2'b11;
          end

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

Num

```
module tb_Num_micro;
    reg [7:0] i_instruccion;
    wire [7:0] Num
    ;

    Num_micro uut
    (
        .i_instruccion(i_instruccion),
        .Num(Num)
        );

        initial
            begin
                i_instruccion = 0;
                #2i_instruccion = 8'b00000001;
                #2i_instruccion = 8'b00001011;
                #2i_instruccion = 8'b11111111;
            end
endmodule
```

Banco de registros

```
module tb_ban_reg_micro;
    reg [5:0] Sel_reg;
    reg W;
    reg [7:0] DW;
    wire [7:0] Rx;
    wire [7:0] Ry;
    reg rst;
    reg clk
   ;

   ban_reg_micro uut
      (
        .Sel_reg(Sel_reg),
        .W(W),
        .DW(DW),
        .Rx(Rx),
        .Ry(Ry),
        .rst(rst),
        .clk(clk)
       );

   initial
      begin
         rst = 1;
```

```
        clk = 0;
        Sel_reg = 0;
        W = 0 ;
        DW = 0;

        #2 rst = 0; Sel_reg = 6'b000_000; W = 0; DW = 8'b00000000;
        #2 Sel_reg = 6'b000000; W = 1; DW = 8'b00001010;
        #2 Sel_reg = 6'b000001; W = 1; DW = 8'b00001011;
        #2 Sel_reg = 6'b000010; W = 1; DW = 8'b00001100;
        #2 Sel_reg = 6'b000011; W = 1; DW = 8'b00001101;
        #2 Sel_reg = 6'b000100; W = 1; DW = 8'b00001110;
        #2 Sel_reg = 6'b000101; W = 1; DW = 8'b00001111;
        #2 Sel_reg = 6'b000110; W = 1; DW = 8'b00010000;
        #2 Sel_reg = 6'b000111; W = 1; DW = 8'b00010001;

        #2 Sel_reg = 6'b001000; W = 0;
        #2 Sel_reg = 6'b011010; W = 0;
        #2 Sel_reg = 6'b101100; W = 0;
        #2 Sel_reg = 6'b111110; W = 0;

        #3 Sel_reg = 6'b000000; W = 1; DW = 8'b00000001;
        #2 Sel_reg = 6'b000001; W = 1; DW = 8'b00000011;
        #2 Sel_reg = 6'b001000; W = 1; DW = 8'b00000100;
    end

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

Alu

```
module tb_alu_micro;
    reg [7:0] Rx;
    reg [7:0] Ry;
    reg [2:0] Sel_op;
    wire[7:0] R0;
    wire [2:0] Ban
    ;

    alu_micro uut
    (
      .Rx(Rx),
      .Ry(Ry),
      .Sel_op(Sel_op),
      .R0(R0),
      .Ban(Ban)
     );

     initial
       begin
           Rx = 76;
           Ry = 44;
           Sel_op = 0;
       #4
           Rx = 76;
```

```
    Ry = 76;
    Sel_op = 1;
#4
    Rx = 76;
    Ry = 80;
    Sel_op = 1;
#6
    Rx = 76;
    Ry = 44;
    Sel_op = 3;
#10
    Rx = 76;
    Ry = 44;
    Sel_op = 4;
#4
    Rx = 76;
    Ry = 44;
    Sel_op = 5;
#4
    Rx = 76;
    Ry = 44;
    Sel_op = 6;
#4
    Rx = 76;
    Ry = 44;
    Sel_op = 7;
end
```

```
endmodule
```

Control

```
module tb_Contro_micro;
    reg [8:0] i_instruccion;
    reg [7:0] Rx;
    reg [2:0] Ban;
    wire [2:0] Sel_op;
    wire [5:0] Sel_reg;
    wire W;
    wire [1:0] Sel_outbus;
    wire [2:0] Sel_DW;
    wire [7:0] o_direccion_instruccion;
    reg rst;
    reg clk;

    Control_micro uut
    (
    .i_instruccion(i_instruccion),
    .Rx(Rx),
    .Ban(Ban),
    .Sel_op(Sel_op),
    .Sel_reg(Sel_reg),
    .W(W),
    .Sel_outbus(Sel_outbus),
    .Sel_DW(Sel_DW),
    .o_direccion_instruccion(o_direccion_instruccion),
```

```
    .rst(rst),
    .clk(clk)
    );

    initial
        begin
        i_instruccion=0;
        Rx=0;
        Ban=0;
        rst=1;
        clk=0;

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

        end

    always
        #1 clk = !clk;

Endmodule
```