Universidad Autónoma de Zacatecas

"Francisco García Salinas"

Programa de Ingeniería en Robótica y Mecatrónica
Unidad Académica de Ingeniería Eléctrica

Materia:

Microcontroladores

Alumnos:

Guadalupe Silva Rodríguez

Juan Francisco Márquez Lamas

Genaro Reyes López

Admilcar Elías Guerrero González

Proyecto:

Microcontrolador

Docente:

Remberto Sandoval Aréchiga

Fecha: 23/11/20

Resumen

En este proyecto se realizó un microprocesador con arquitectura Harvard con ayuda de la metodología aprendida en el curso la cual consistió en realizar una serie de pasos determinados para llevar un orden y realizar de manera satisfactoria un microprocesador. Primero se realizó el diagrama de caja negra con el cual se dio una idea de cómo funcionaría dicho proyecto, posteriormente se creó un diagrama general con todos los bloques que constituyen al microprocesador (diagrama de caja blanca), de ahí se fue centrando en cada bloque por separado y se creó una serie de decodificaciones para las instrucciones y señales, una vez clara la función de todas nuestras señales y como se comportaban, se procedió al pseudocódigo de cada bloque, de esta manera es más sencillo terminar el código final el cual consistió solo en unir todas las partes en el software de vivado donde se realizaron las pruebas en base al lenguaje de descripción de hardware verilog.

Índice

Introducción

Alguna vez te has cuestionado ¿Qué es un microprocesador? En general, el término "microprocesador", remite al dispositivo principal dentro de las computadoras digitales, es decir, al elemento encargado de realizar los cálculos que permiten desde escribir una carta hasta editar una fotografía; desde administrar una nómina hasta platicar en tiempo real con alguna persona al otro lado del mundo; desde disfrutar una película hasta controlar complejos procesos industriales; en fin, el concepto de "microprocesador" evoca un dispositivo de enorme poder de cálculo, relativamente costoso, que consume mucha potencia y que, por tanto, sólo está al alcance de pocas personas. Sin embargo, la realidad es muy distinta. En un hogar típico, existen una enorme cantidad de microprocesadores, realizando diversas tareas que buscan facilitar la vida diaria de los usuarios. Se encuentran en el televisor, en el equipo de sonido, en el reproductor de DVD, en los teléfonos celulares, y es que esta tecnología se ha abaratado a tal grado que muchas aplicaciones que antes requerían el uso de varios dispositivos individuales, ahora se pueden realizar con más facilidad y de manera más económica con la aplicación de un microprocesador, o de su variante, un microcontrolador.

> Sabías que...
>
> El primer microprocesador fue el 4004 de Intel, y apareció en el mercado en 1971. Este hecho fue accidental y se debió a un contrato entre la empresa Intel y una compañía japonesa de calculadores para el desarrollo de un circuito integrado para dicho producto. El desarrollo no fue aceptado y la firma Intel se decidió por su comercialización. El 4004 era un microprocesador de 4 bits realizado con tecnología PMOS.

Esto significa que los microprocesadores y microcontroladores se han convertido en parte de la vida diaria, y esto a su vez implica que cualquier persona interesada en el área de la electrónica o del control debe saber cómo funcionan y cómo se aplican estos dispositivos, es por esto que en este proyecto se dio a la tarea de conocer,

Figura 1 Arquitectura Harvard

investigar y crear un microcontrolador en base a una arquitectura Harvard, este modelo, es utilizado por los Microcontroladores PIC, tiene la unidad central de proceso (CPU) conectada a dos memorias (una con las instrucciones y otra con los datos) por medio de dos buses diferentes. Una de las memorias contiene solamente las instrucciones del programa (Memoria de Programa), y la otra sólo almacena datos (Memoria de Datos).

Con esto en mente es que se realizó el microprocesador el cual se presenta a continuación. Se espera un buen recibimiento.

Requerimientos

Se requiere un sistema que sea capaz de realizar los procesos necesarios para construir un microcontrolador de 8 bits guiado por un set de instrucciones y una metodología preestablecida. Dicho sistema debe permitir que los datos que entren puedan ser procesados y manejados a través de éste. El sistema debe contar con dos entradas que indiquen las instrucciones (9 bits) y la entrada de datos (8 bits), asimismo cuenta con cuatro salidas, las cuales definen la dirección de instrucciones (8 bits), la dirección de datos (8 bits), el bus de datos (8 bits) y el bus RW (define lectura o escritura de 1 bit). Adicionalmente se requieren que dicho sistema esté basado en la arquitectura Harvard, conociendo la diferencia entre otras arquitecturas así como sus ventajas y desventajas de la misma.

Arquitectura

Caja negra. En la *Figura 2* se muestra la caja negra del microprocesador creado. Así como en las *Tabla 1 y 2* la descripción de las entradas y salidas.

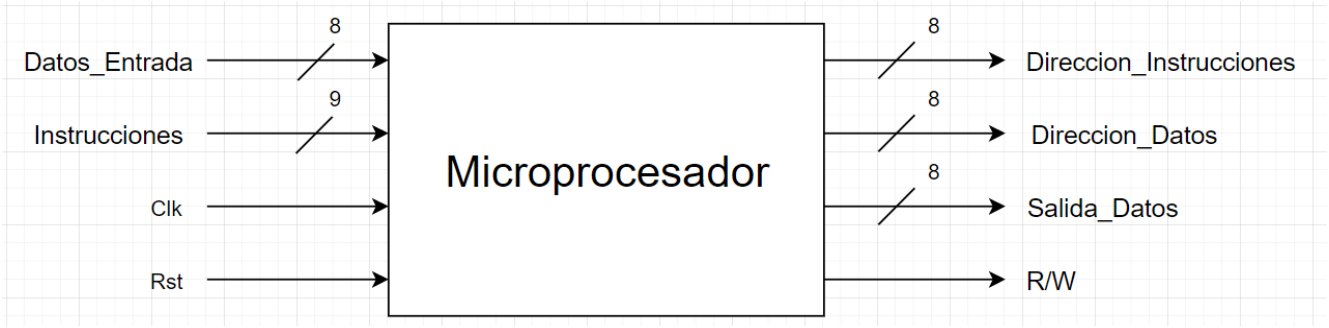

Figura 2 Caja negra Microprocesador

Tabla 1 Entradas Microprocesador

ENTRADAS	TAMAÑO (BITS)	DESCRIPCION
Datos_Entrada	8	Lleva los datos que entran al microprocesador sobre los que se realizan las instrucciones, los cuales son guardados de forma temporal en los registros.
Instrucciones	9	Lleva las instrucciones que se van a realizar dentro del procesador, según el set de instrucciones.
Clk	1	Señal referente al tiempo, es una señal periódica con frecuencia de 100MHz
Rst	1	Señal que reinicia el microprocesador configurándolo a un estado inicial.

Tabla 2 Salidas Microprocesador

SALIDAS	TAMAÑO (BITS)	DESCRIPCION
Direccion_Instrucciones	8	Controla las direcciones de la memoria de instrucciones.
Direccion_Datos	8	Controla las direcciones de memoria , de la memoria de datos
Salida_Datos	8	Lleva los datos que salen del microprocesador y que serán almacenados en la memoria de datos.
R/W	1	indicador de acción en la memoria, referente a lectura o escritura

Estructura Harvard. En la *Figura 3* se muestra el diagrama del micro controlador creado con la arquitectura Harvard. Así como en las *Tabla 3* la descripción de los bloques.

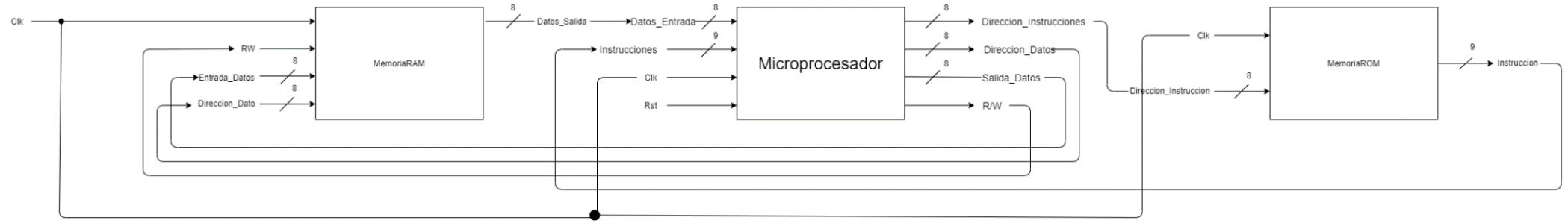

Figura 3 Diagrama Estructura de Harvard

Tabla 3 Descripción diagrama de Harvard

BLOQUE	DESCRIPCIÓN
Memoria RAM	Memoria de almacenamiento donde se guardan los datos procesados o a procesar por el microprocesador, es una memoria de lectura y escritura de 2^8 celdas de almacenamiento
Microprocesador	Memoria de solo lectura de 2^8 × 9 o 256×9 donde se almacena la secuencia de instrucciones que serán realizadas por el microprocesador
Memoria ROM	Sistema de procesamiento que permite realizar operaciones de acuerdo a un set de instrucciones establecido

Caja blanca. En la *Figura 4* se muestra la caja blanca del microprocesador creado. Así como en la *Tabla 4* la descripción de las entradas y salidas.

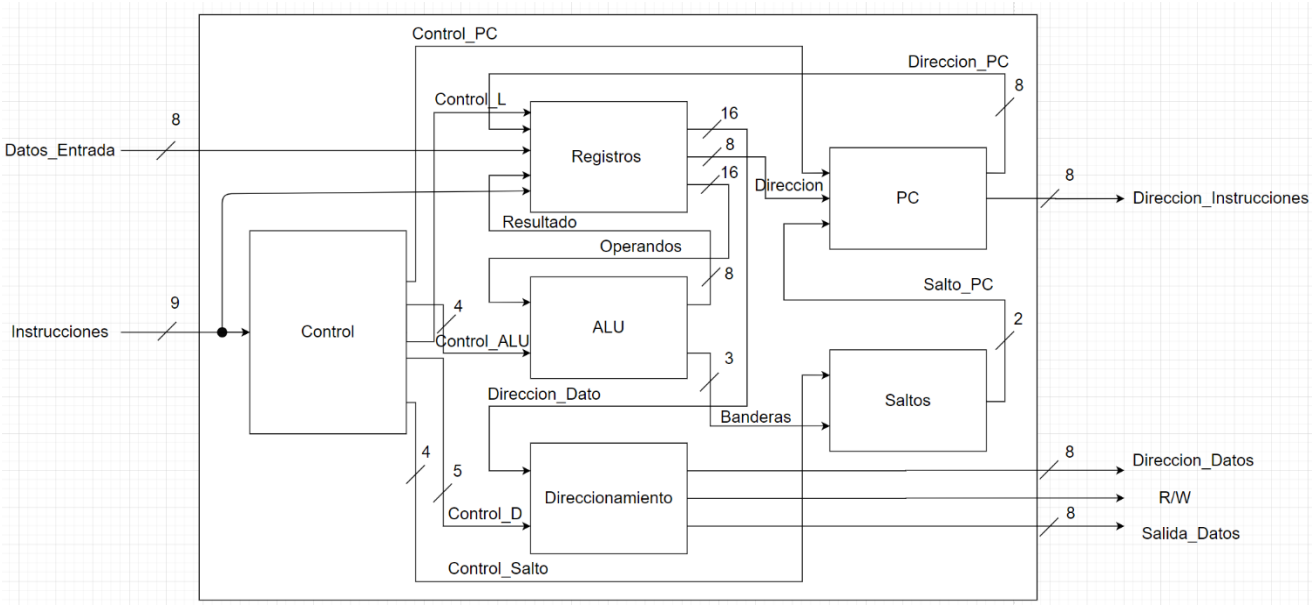

Figura 4 Caja blanca

Tabla 4 Caja blanca

BLOQUE	DESCRIPCIÓN
Control	Sincroniza y procesa las instrucciones. Coordina todos los componentes del microprocesador, lo que hace que cada instrucción siga una secuencia apropiada en el momento correcto. Decodifica y/o entiende las instrucciones que provienen de la memoria de instrucciones, y dirige la acción para realizarlas.
Registros	Contiene registros de propósito general, en los que se guardan datos temporales sobre los que se realizan las instrucciones (es decir, los datos u operandos para realizar las operaciones del set de instrucciones).
ALU	Realiza las operaciones aritméticas y lógicas. Del set de instrucciones se refiere a la instrucción MATH y las operaciones que es capaz de realizar son en este caso: suma y resta sobre enteros, corrimientos a derecha e izquierda, negación, AND, OR Y XOR.
Direccionamiento	Controla la memoria de datos, coordinando el bus de dirección a memoria (Direccion_Datos), el bus de salida de datos (Salida_Datos) y el indicador de acción lectura o escritura (R/W).
Saltos	Se encarga de realizar los saltos del PC trátese de saltos condicionales o no condicionales.
PC	Lleva la dirección de memoria donde se guarda la siguiente instrucción a realizar.

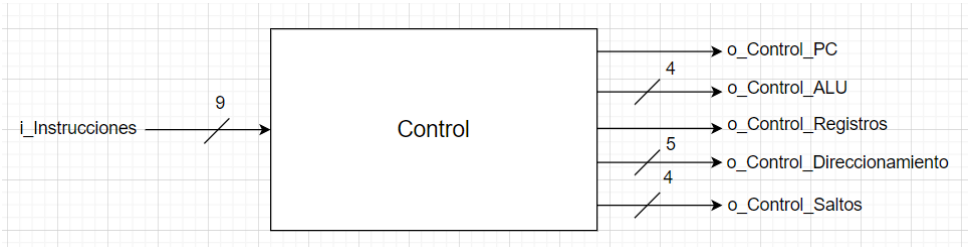

Figura 5 Bloque control

Control. En la *Figura 5* se muestra el bloque control del microprocesador creado. Así como en las *Tabla 5 y 6* la descripción de las entradas y salidas.

Tabla 5 Entradas Control

ENTRADAS	TAMAÑO (BITS)	DESCRIPCION
i_Instrucciones [8:6]	3	Para la segmentación de los bits, los bits más significativos corresponden a la instrucción que se realizará.
i_Instrucciones [5:0]	6	Los bits restantes se agrupan igualmente en segmentos de 3 bits y según la instrucción de la que se trate el cada grupo corresponde a los argumentos de la instrucción. Por ejemplo, para la primera instrucción los bits menos significativos corresponden a la variable #NUM, y los más significativos (que en todo el código de instrucción serían los intermedios [5:3]) corresponden al registro donde se carga el dato

Tabla 6 Salidas Control

SALIDAS	TAMAÑO (BITS)	DESCRIPCION
o_Control_PC	1	Actualiza los datos del PC incrementando a 1 para así obtener la siguiente instrucción a realizar.
o_Control_ALU	4	La señal lleva el código de la operación a realizar correspondiente al argumento OP de la instrucción MATH en los bits menos significativos ([2:0]) y el bit más significativo corresponde a la instrucción, es decir, cuando este bit valga 1 entonces se realizará la operación, de lo contrario la ALU no realizará ninguna operación
o_Control_Direccionamiento	5	Salida dirigida a direccionamiento la cual lleva el código de la instrucción que se va a ejecutar y el código del dato que se va a

		almacenar cuando se trata de un direccionamiento inmediato.
O_Control_Saltos	4	Señal que controla la ejecución de saltos, lleva el código de la condición que se está revisando [2:0] y el código de instrucción para verificar si va a realizar la acción o no.
O_Control_Registros	1	Señal dirigida a registros, la cual controla la carga de los datos provenientes de PC y ALU.

Descripción funcional

Pseudocódigo:

```
I/O
[8:0] i_Instrucciones;
i_Rst;
i_Clk;
o__Control_PC;
[2:0]o_Control_ALU;
[4:0]o_Control_Direccionamiento;
[3:0]o_Control_Saltos;
o_Control_Registros;

segmentar i_Instrucciones:
c_in <-- i_Instrucciones ([8:6]);
c_reg <-- i_Instrucciones ([5:3]);
c_arg <-- i_Instrucciones ([2:0]);

INICIO

si hay un flanco positivo de reloj:

if (Rst = 1) then
o_Control_PC <= 0;
o_Control_ALU <=0;
o_Control_Registros <= 0;
o_Control_Direccionamiento <=0;
o_Control_Saltos <=0;

else

case (c_in)
001: o_Control_Direccionamiento <= {00,c_arg}; o_Control_ALU <= {0,c_arg}; o_Control_Saltos <= {0,
c_arg};
      o_Control_PC  <=1; o_Control_Registros <= x;

010: o_Control_Direccionamiento <= {01,c_arg}; o_Control_ALU <= {0,c_arg}; o_Control_Saltos <= {0,
c_arg};
      o_Control_PC <=1; o_Control_Registros <= x;

011: o_Control_Direccionamiento <= {10,c_arg}; o_Control_ALU <= {0,c_arg}; o_Control_Saltos <= {0,
c_arg};
      o_Control_PC <=1; o_Control_Registros <= x;

100: o_Control_Direccionamiento <= {11,c_arg}; o_Control_ALU <= {0,c_arg}; o_Control_Saltos <= {0,
c_arg};
      o_Control_PC <=1; o_Control_Registros <= x;
```

```
101: o_Control_Direccionamiento <= {00,c_arg}; o_Control_ALU <= {0,c_arg}; o_Control_Saltos <= {0,
c_arg};
      o_Control_PC <=1; o_Control_Registros <= x;

110: o_Control_Direccionamiento <= {00,c_arg}; o_Control_ALU <= {1,c_arg}; o_Control_Saltos <= {0,
c_arg};
      o_Control_PC <=1; o_Control_Registros <= 0;

111: o_Control_Direccionamiento <= {00,c_arg}; o_Control_ALU <= {0,c_arg}; o_Control_Saltos <= {1,
c_arg};
      o_Control_PC <=0; o_Control_Registros <= 1;

default: o_Control_Direccionamiento <= o_Control_Direccionamiento; o_Control_ALU <=
o_Control_ALU;
          o_Control_Saltos <= o_Control_Saltos; o_Control_PC <=1; o_Control_Refistros <= x;
endcase

FIN
```

Registros. En la *Figura 6* se muestra el bloque control del microprocesador creado. Así como en las *Tabla 7* la descripción de las entradas y salidas.

Figura 6 Bloque Registros

Tabla 7 Entradas de Registros

ENTRADAS	TAMAÑO (BITS)	DESCRIPCION
i_Datos_Entrada	8	Trae los datos que se cargarán en los registros provenientes de la memoria de datos, los cuales son leídos cuando se ejecuta la instrucción 2
i_Resultado_ALU	8	Entrada que proviene de la ALU, la cual trae el resultado de la operación que previamente realizó la ALU y que será acumulada en el registro R0.
i_Intrucciones	9	Entrada proveniente de la memoria de instrucciones, la cual trae el código de la instrucción que se va a ejecutar [8:6], el código del registro que se va a manipular [5:3] y el código del dato que se va a cargar [2:0], en esta caso para la primera instrucción. para cada una de las instrucciones los bits [5:0] corresponden a los argumentos de la instrucción.
i_Direccion_PC	8	Entrada proveniente de PC, la cual trae el valor actual de PC la cual será guardada en el registro R7 cuando se realice la condición 1 correspondiente a los saltos.
i_Clk	1	Señal referente al tiempo, es una señal periódica con frecuencia de 100MHz
i_Rst	1	Señal que restablece el sistema en una configuración inicial
I_Control_Registros	1	Señal proveniente de control con la que se controla la carga de los valores provenientes de PC y ALU respectivamente.

Tabla 8 Salidas Registros

SALIDAS	TAMAÑO (BITS)	DESCRIPCION
o_Operandos	16	Salida dirigida a la ALU, donde se llevan los datos para que esta realice alguna operación, los cuales se segmentan de la siguiente forma: los bits menos significativos [7:0] corresponden al dato en registro R0 y los bits más significativos [15:8] corresponden al dato guardado en el registro RX
o_DireccionDato	16	Salida dirigida a direccionamiento, la cual lleva el dato que se va a almacenar en la memoria y la dirección donde se va a realizar la carga, segmentados de tal forma que los bits más significativos corresponden a la dirección de memoria y los bits menos significativos corresponden al dato que se va a almacenar.
o_Direccion_Salto	8	Salida de dirigida a PC la cual lleva la dirección de memoria a la que va a saltar PC si se cumple una condición.

Descripción funcional

```
Pseudocódigo:

I/O
i_Rst;
i_Clk;
i_Direccion_PC;
i_Resultado_ALU;
i_Control_Registros;
i_Datos_Entrada;
i_Instrucciones;
o_Direccion_Salto;
o_Operandos;
o_DireccionDato;

señales:
c_ins <= i_Instrucciones[8:6];
c_RX <= i_Instrucciones[5:3];
c_RY <= i_Instrucciones[2:0];

reg[7:0] BancoRegistros [0:7];
interger i;

INICIO

si hay un flanco positivo de reloj:
si (i_Rst = 1) entonces
inicia ciclo for:
for(i = 0; i <8; i = i+1)
BancoRegistros[i] <= 0;
```

```
termina el ciclo for

de lo contrario:

case(i_Ins)
001: BancoRegistros [c_RX] <= {00000, c_RY};
010: o_DireccionDato <= { BancoRegistros[c_RY], BancoRegistros[c_RX]};
        if  (i_Clk == 1) entonces: BancoRegistros[c_RX] <= i_Datos_Entrada;

011: o_DireccionDato <= {BancoRegistros[c_RX], BancoRegistros[c_RY]};
100: o_DireccionDato <= { BancoRegistros[c_RX], BancoRegistros [c_RY];
101: BancoRegistros [c_RX] <= BancoRegistros[c_RY];
110: o_Operandos <= { BancoRegistros[c_RX], Banco_Registros[000] };
111: o_Salto_PC <= BancoRegistros[c_RX];
default: // sin operacion o_DireccionDatos <= o_DireccionDato; o_Salto_PC <= o_Salto_PC;
o_Operandos <= o_Operandos;
termina case

si(i_control_Registros =0) entonces
BancoRegistros[111] <= i_Direccion_PC;

de lo contrario:
BancoRegistros[000] <= i_Resultado_ALU;

termina bloque de reloj

FIN
```

ALU. En la Figura 7 se muestra el bloque ALU. En las tablas 9 y 10 se muestra la descripción de las entradas y salidas.

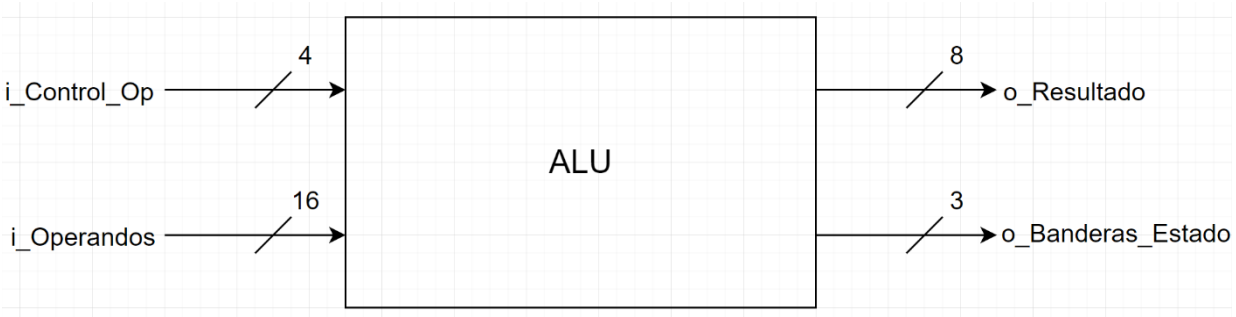

Figura 7 Bloque ALU

Tabla 9 Entradas del bloque ALU

ENTRADAS	TAMAÑO (BITS)	DESCRIPCION
i_Control_Op	4	Entrada proveniente de control, la cual contiene el código de la operación a realizar en los bits menso significativos [2:0], y en el bit más significativo corresponde a la instrucción [3] si este vale 1 entonces la ALU realiza la instrucción de lo contrario no ejecuta nada.
i_Opreandos	16	Entrada procedente de registros, la cual trae los datos de los operandos necesarios para realizar la operación, donde los bits menos significativos [7:0] corresponden al registro R0 y los bits más significativos [15:0] corresponden al registro RX.

Tabla 10 Salidas del bloque ALU

SALIDAS	TAMAÑO (BITS)	DESCRIPCION
o_Resultado	8	Salida dirigida a registros para guardar el resultado que se obtuvo de la operación realizada, la cual será almacenada en R0.
o_Banderas_Estado	3	Salida dirigida a saltos para informar si se levantó alguna bandera de condición.

Descripción funcional

Pseudocódigo:

```
I/O
[3:0] i_Control_ALU;
[15:0] i_Operandos;
[7:0] o_Resultado;
[2:0 ]o_Banderas_Estado;

señales internas
reg Z;
reg N;
reg C;
reg signed [8:0]Resultado;


INICIO

SEGMENTAR:
Operacion <-- i_Control_ALU([2:0]);
Control<-- i_Control_ALU([3]);

reg signed op2 <-- i_Operandos([15:8]);
reg signed op1 <-- i_Operandos([7:0]);

if (Control)
 case (Operacion)
000 : Resultado <-- op1+op2;
001 : Resultado <-- op1-op2;
010 : Resultado <-- op1<<op2;
011 : Resultado <-- op1>>op2;
100 : Resultado <-- not(op2);
101 : Resultado <-- op1 & op2;
110 : Resultado <-- op1 | op2;
default : Resultado <-- op1 ^ op2;
endcase

assign Z = (Resultado == 0) ? 1´b1 : 1´b0;
assign N = (Resultado[8] == 1) ? 1´b1 : 1´b0;
assign C = (Resultado[8] == resultado[7]) ? 1´b0: 1´b1;

o_Banderas_Estado <= { Z, N, C };
o_Resultado <= Resultado[7:0];

FIN
```

Direccionamiento. En la *Figura 8* se muestra el bloque control. Así como en las *Tabla 11 y 12* la descripción de las entradas y salidas.

Figura 8 Bloque direccionamiento

Tabla 11 Entradas Direccionamiento

ENTRADAS	TAMAÑO (BITS)	DESCRIPCION
i_DireccionDato	16	Entrada proveniente de registros la cual trae el código del dato que se va almacenar en la memoria, así como la dirección donde se realizará el almacenamiento, donde los bits más significativos corresponden a la dirección de memoria y los menos significativos corresponden al código del dato que se va a almacenar.
i_Control_Direc	5	Entrada procedente de control, la cual trae el código de la instrucción que se desea ejecutar y el valor del dato que se va almacenar en la memoria cuando se trata de un modo de direccionamiento inmediato.

Tabla 12 Salidas Direccionamiento

SALIDAS	TAMAÑO (BITS)	DESCRIPCION
o_Direccion_Datos	8	Salida dirigida a la memoria de datos donde se lleva la dirección de memoria donde se va a almacenar el dato.
o_Salida_Datos	8	Salida dirigida a la memoria de datos donde se lleva el dato que se va a almacenar.
o_R/W	1	Salida dirigida a la memoria para indicar si la acción que se realiza en esta es de lectura o escritura

Descripción funcional

Pseudocódigo:

I/O

[4:0] i_Control_Direc;
[15:0] i_DireccionDato;
[7:0] o_Direccion_Datos;
[7:0] o_Salida_Datos;
R/W;

INICIO

//segmentar:
señales internas
[1:0]c_in <= i_Control_Direc[4:3];
[2:0]c_dat <= i_Control_Direc[2:0];

case (c_in)
00: // no hay operación; o_Direccion_Datos <= z; o_Salida_Datos <= z; R/W <= z;
01: o_Direccion_Datos <= i_Registros[15:8]; R/W <= 1; // lectura
10: o_Direccion_Datos <= i_Registros[15:8]; o_Salida_Datos <= {00000, c_dat}; R/W <= 0;
11: o_Direccion_Datos <= i_Registros[15:8]; o_Salida_Datos <= i_Registros[7:0]; R/W <= 0;
endcase

FIN

Saltos. En la *Figura 9* se muestra el bloque control del microprocesador creado. Así como en las *Tabla 13 y 14* la descripción de las entradas y salidas.

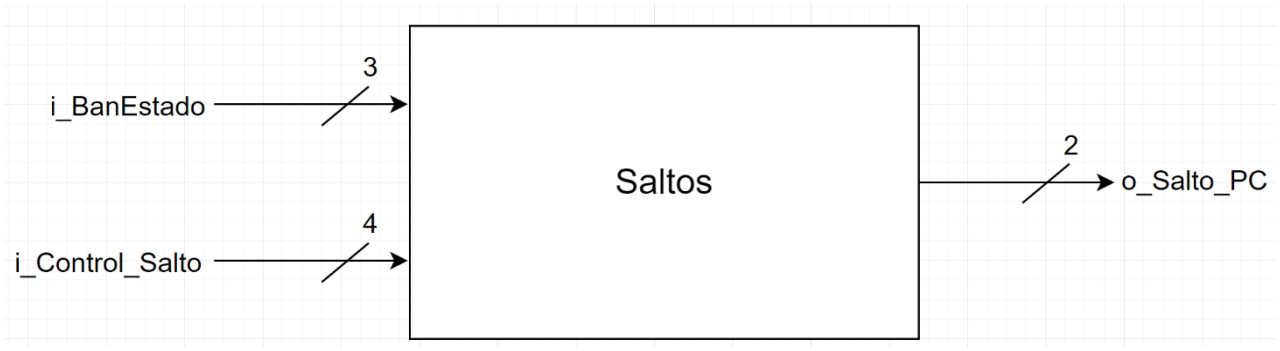

Figura 9 Bloque Saltos

Tabla 13 Entradas Saltos

ENTRADAS	TAMAÑO (BITS)	DESCRIPCION
i_BanEstado	3	Entrada proveniente de la ALU la cual indica si se ha levantado alguna bandera.
i_Control_Salto	4	Entrada procedente de control, la cual trae el código de la condición que se desea verificar [2:0] y la indicación para que el bloque realice la instrucción [3].

Tabla 14 Salidas Saltos

SALIDAS	TAMAÑO (BITS)	DESCRIPCION
o_Salto_Pc	2	Salida dirigida a PC para indicar que se va a realizar un salto. Donde las indicaciones se presentan de la siguiente forma: cuando: se cumple una condición o_PC vale 01. se cumple la condición 1 (sin condicion a R7), o_PC vale 10 no se cumple la condición que se evalúa o_PC vale 00

Descripción funcional

Pseudocódigo:

```
I/O

   [3:0] i_Control_Salto;


   [2:0] i_BanEstado;

   [1:0] o_Salto_PC;


INICIO

si (i_Control_Salto[3] = 1) entonces
   Case (i_Control_Salto[2:0])
      000: o_PC <= 01;


      001: o_PC <= 10;

      010:

         si (i_BanEstado(2) = 1) entonces
            o_Salto_PC <= 01;
         de lo contrario
            o__Salto_PC <= 00;

      011:
         si(i_BanEstado(2) = 0) entonces
            o_Salto_PC <= 01;
         de lo contrario


            o_Salto_PC <= 00;

      100:

         si (i_BanEstado(1) = 1) entonces

            o_Salto_PC <= 01;


         de lo contrario

            o_Salto_PC <= 00;

      101:
```

```
        si(i_BanEstado(1) = 0) entonces

            o_Salto_PC <= 01;


        de lo contrario

            o_Salto_PC <= 00;

      110:
        si(i_BanEstado(0) = 1) entonces
          o_Salto_PC <= 01;


        de lo contrario


          o_Salto_PC <= 00;


      111:
        si (i_BanEstado(0) = 0) entonces
           o__Salto_PC <= 01;
        de lo contrario
           o_Salto_PC <= 00;
termina case

si (i_Control_Salto[3] == 0 ) entonces
       o_Salto_PC <= 11;

FIN
```

PC. En la *Figura 10* se muestra el bloque control del microprocesador creado. Así como en las *Tabla 15 y 16* la descripción de las entradas y salidas.

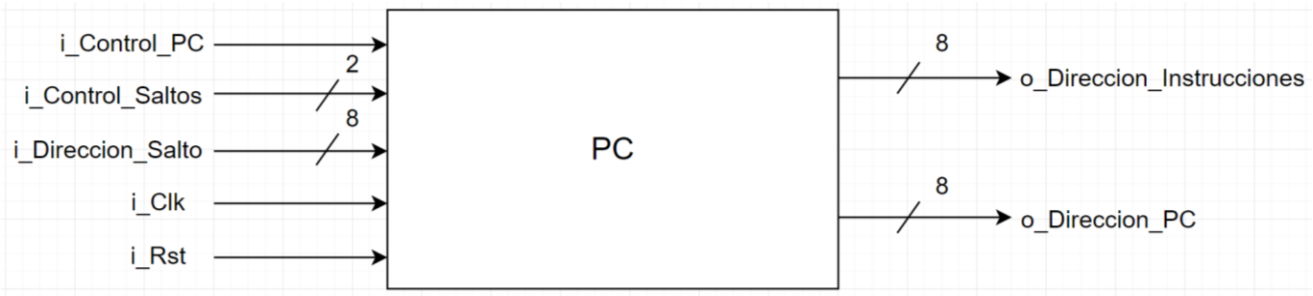

Figura 10 Bloque PC

Tabla 15 Entradas PC

ENTRADAS	TAMAÑO (BITS)	DESCRIPCION
i_Control_Pc	1	Entrada proveniente de control la cual indica que el PC debe actualizar su valor con un incremento de 1.
i_Control_Saltos	2	Entrada proveniente de saltos la cual indica que se debe realizar un salto en el PC porque se ha cumplido una condición.
i_Direccion_Salto	8	Entrada proveniente de registros donde se específica la dirección de memoria a la que va a saltar el PC.
i_Clk	1	Señal referente al tiempo, es una señal periódica con frecuencia de 100MHz
i_Rst	1	Señal que restablece el sistema en una configuración inicial

Tabla 16 Salidas PC

SALIDAS	TAMAÑO (BITS)	DESCRIPCION
o_Direccion_Instrucciones	8	Salida dirigida a la memoria de instrucciones la cual controla las direcciones de memoria donde se almacena la siguiente instrucción a realizar.
o_Direccion_PC	8	Salida dirigida a los registros la cual lleva el valor actual del PC que se va a almacenar en R7 cuando se cumpla la condición 1 de la instrucción saltos.

Implementación y simulación

Simulaciones

Enseguida presentamos en la ilustración 11 la simulación del micro controlador donde se implementamos la arquitectura Harvard, así mismo incluimos en las ilustraciones 12 y 13 las simulaciones de las memorias RAM y ROM, también se incluyó la simulación del microprocesador en las figuras 14 y 15, por último en las ilustraciones de la 16 a la 21 son las simulaciones de cada bloque de la estructura del microprocesador diseñado.

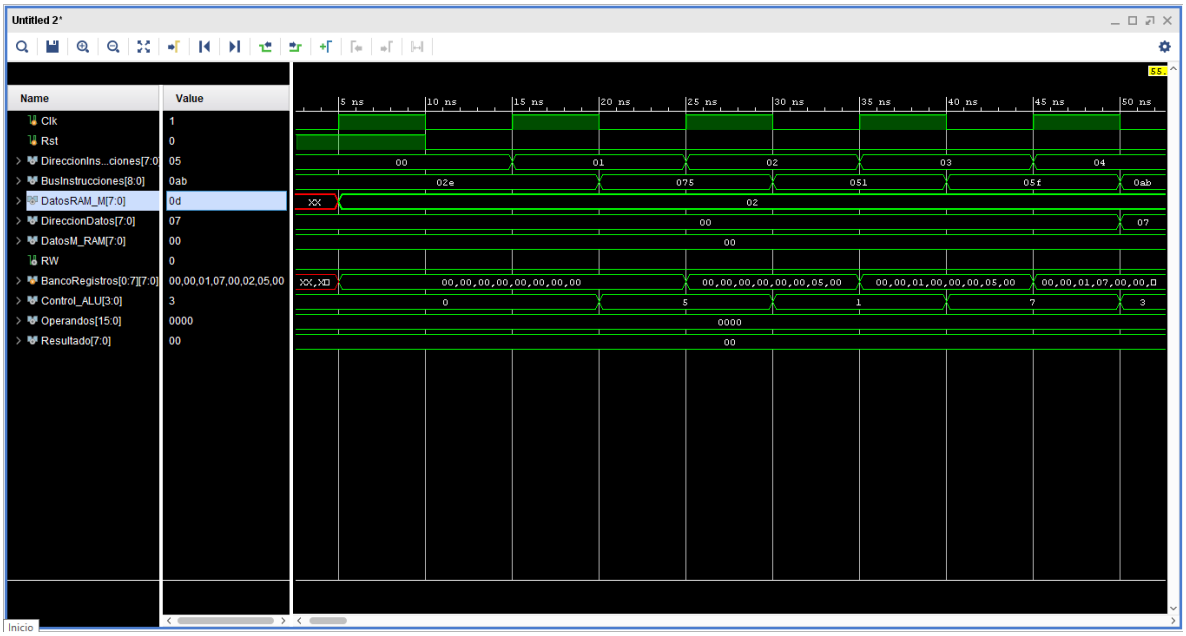

Figura 11 Simulación Micro controlador

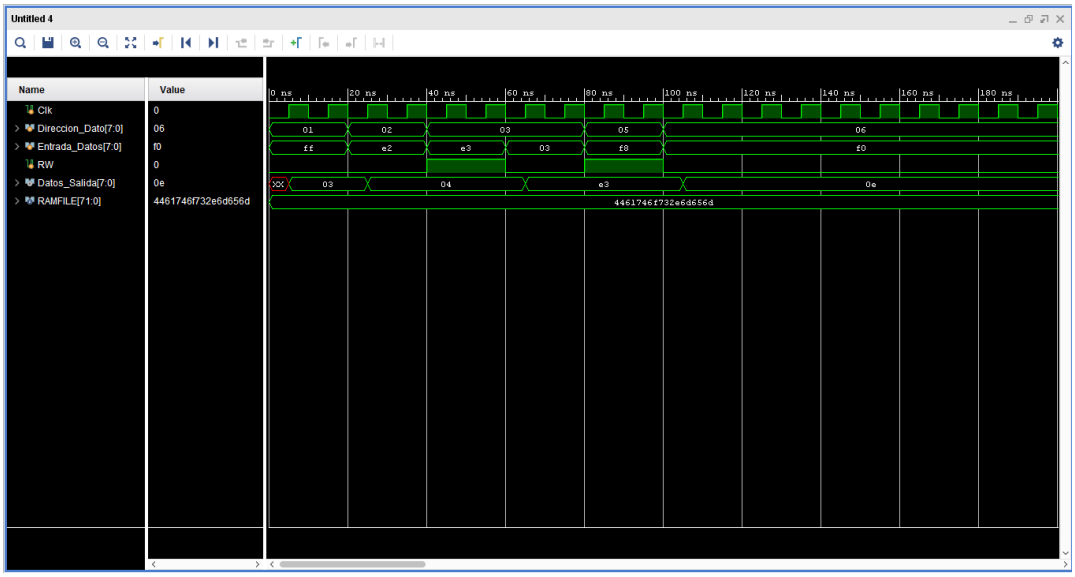

Figura 12 Simulación memoria RAM

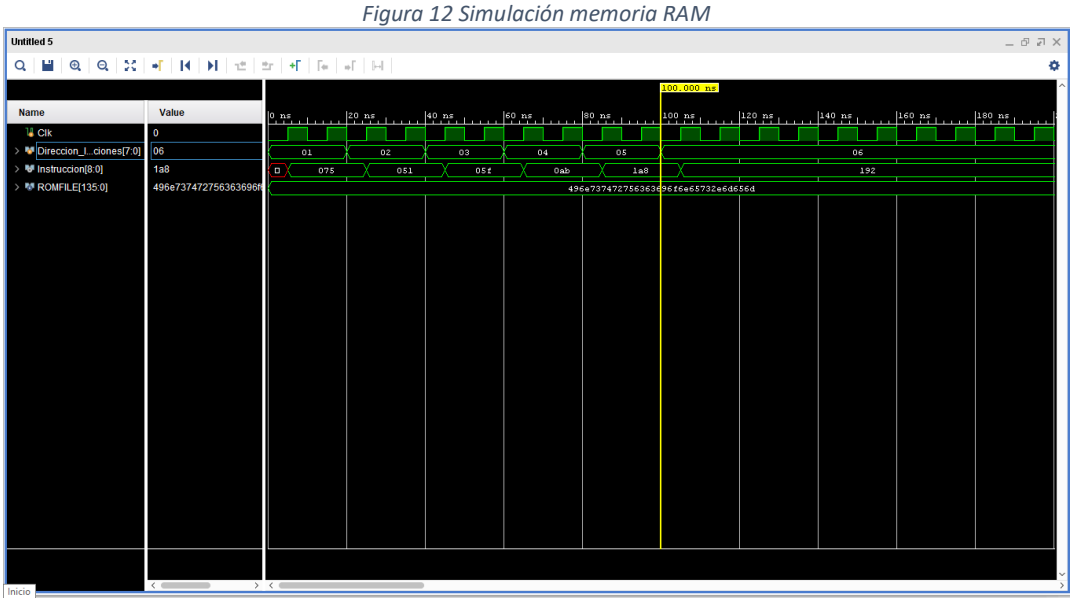

Figura 13 Simulación memoria ROM

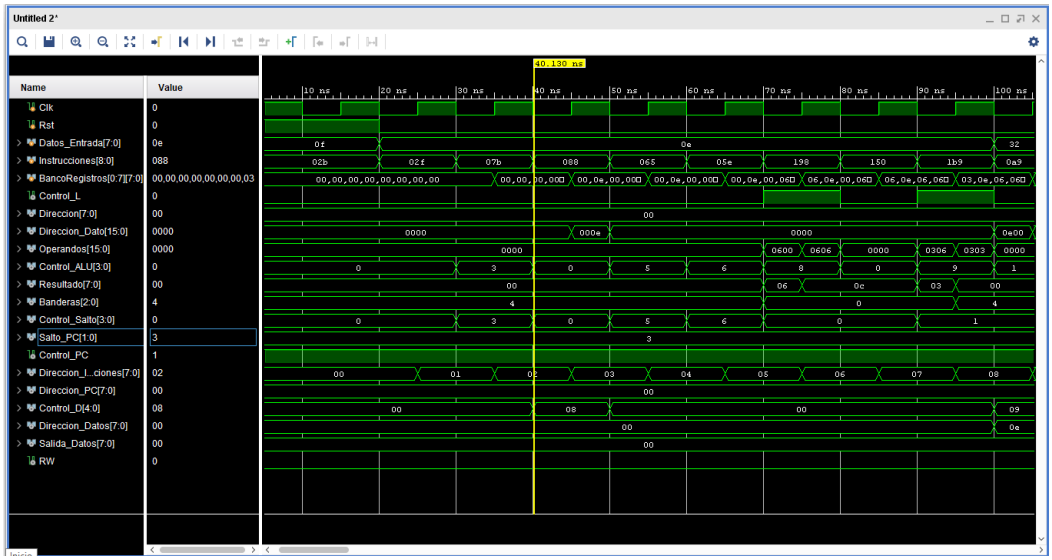

Figura 14 Simulación microprocesador

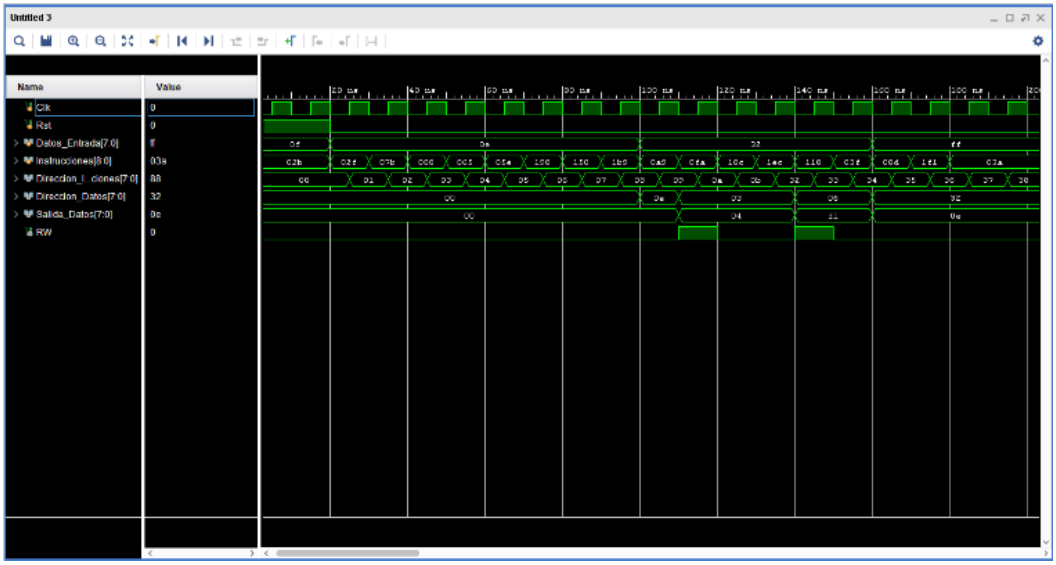

Figura 15 Simulación microprocesador

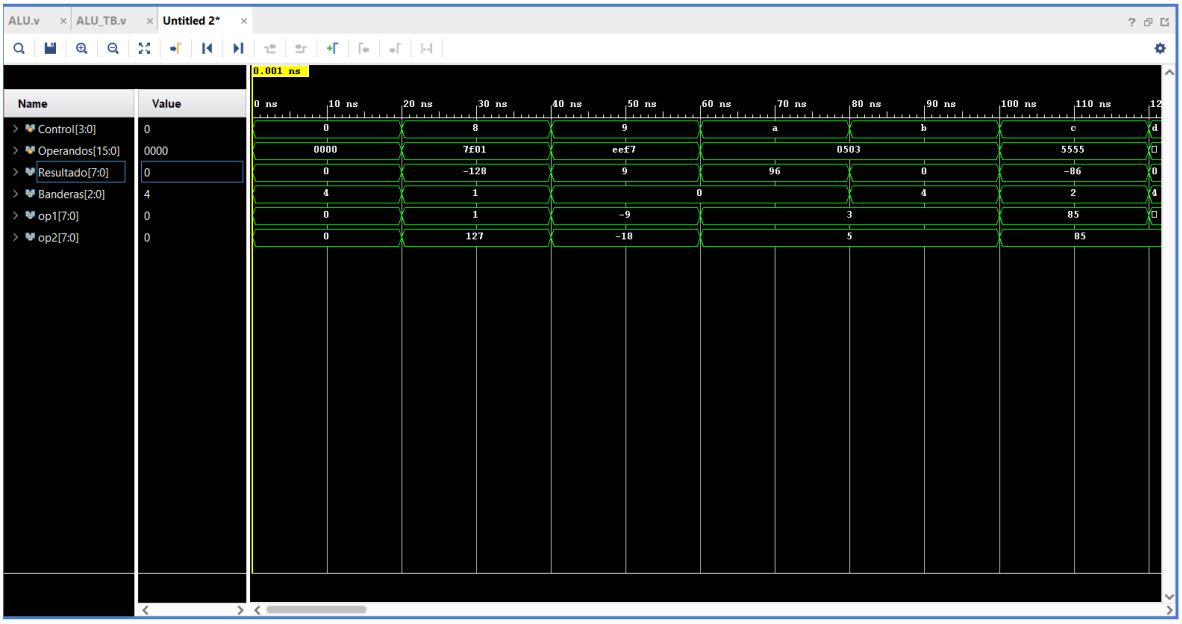

Figura 16 Simulación ALU

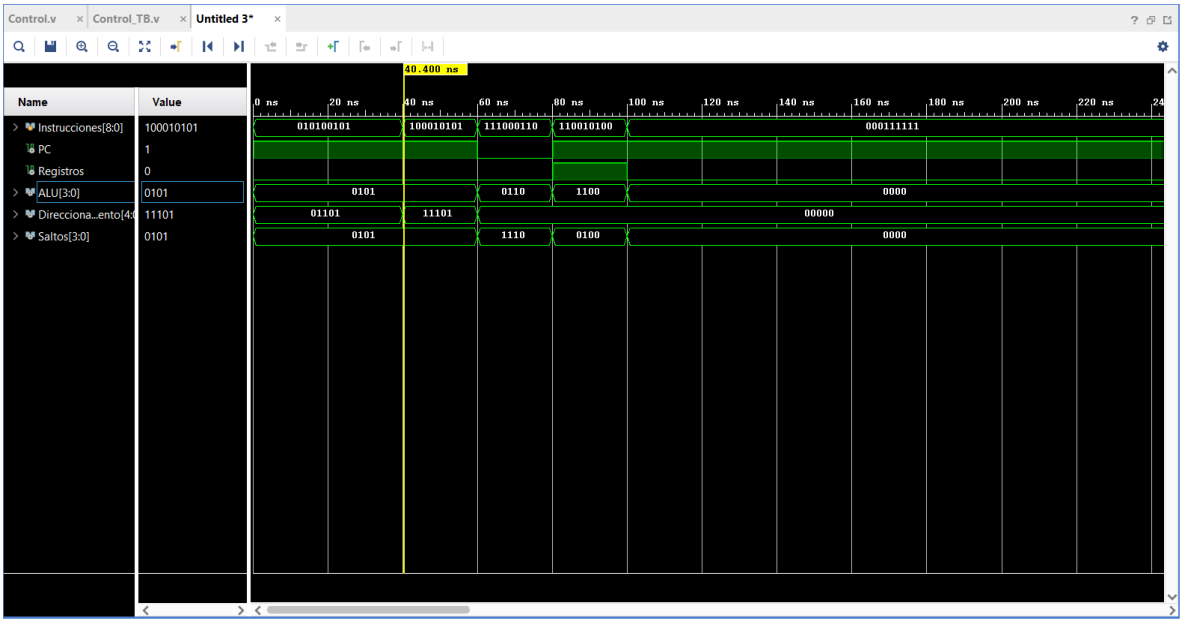

Figura 17 Simulación Control

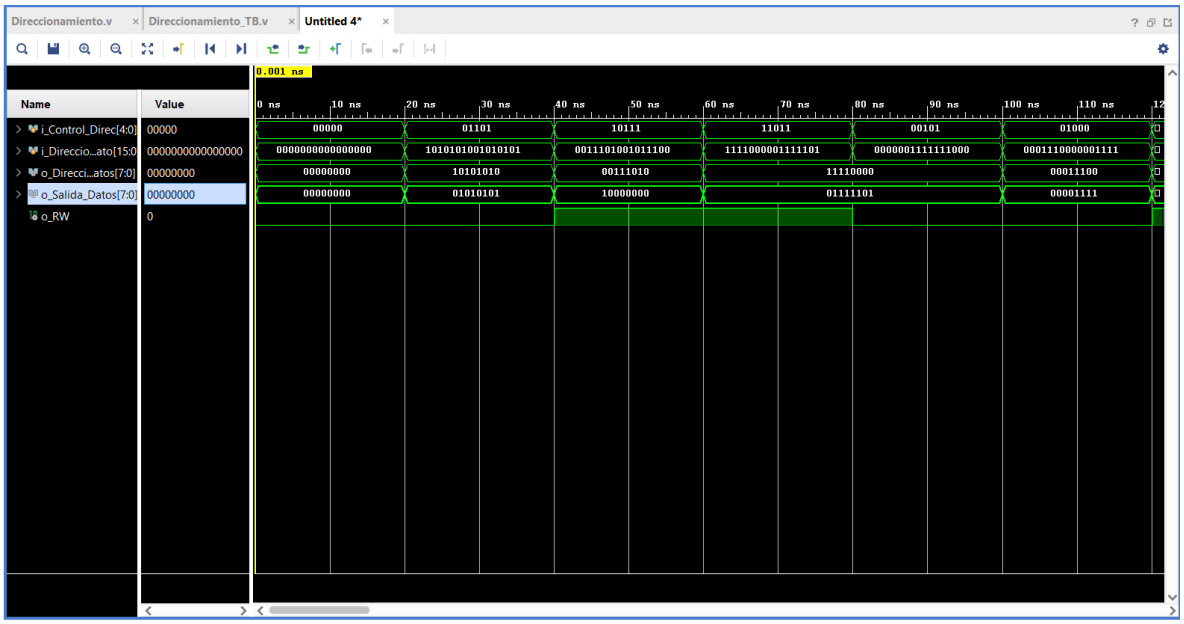

Figura 18 Simulación Direccionamiento

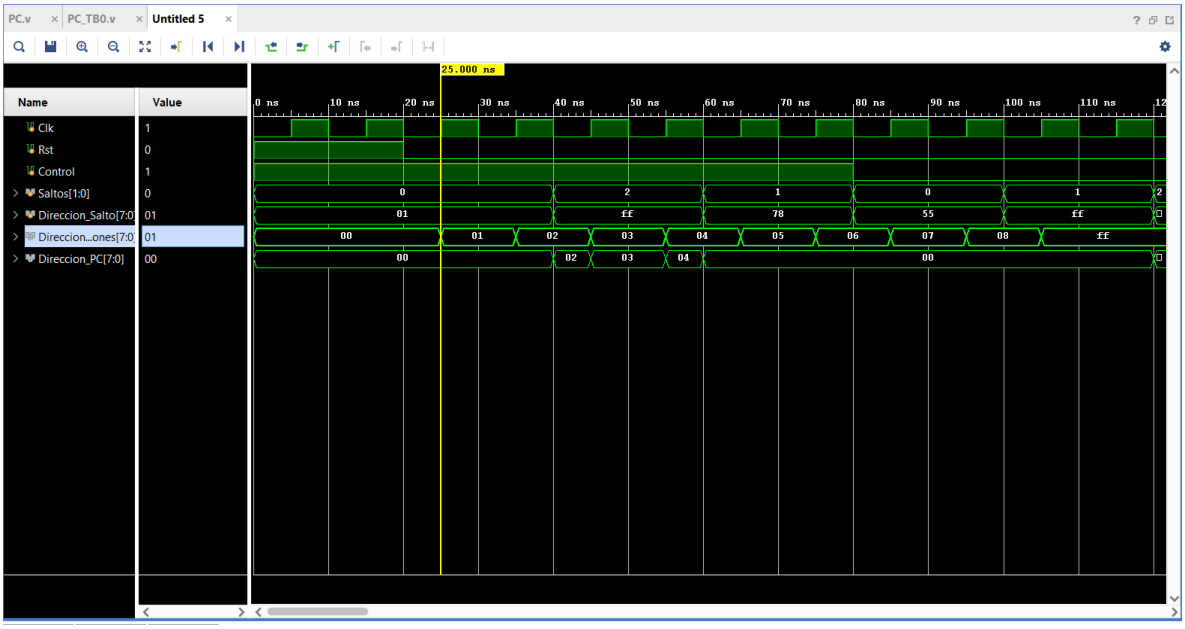

Figura 19 Simulación PC

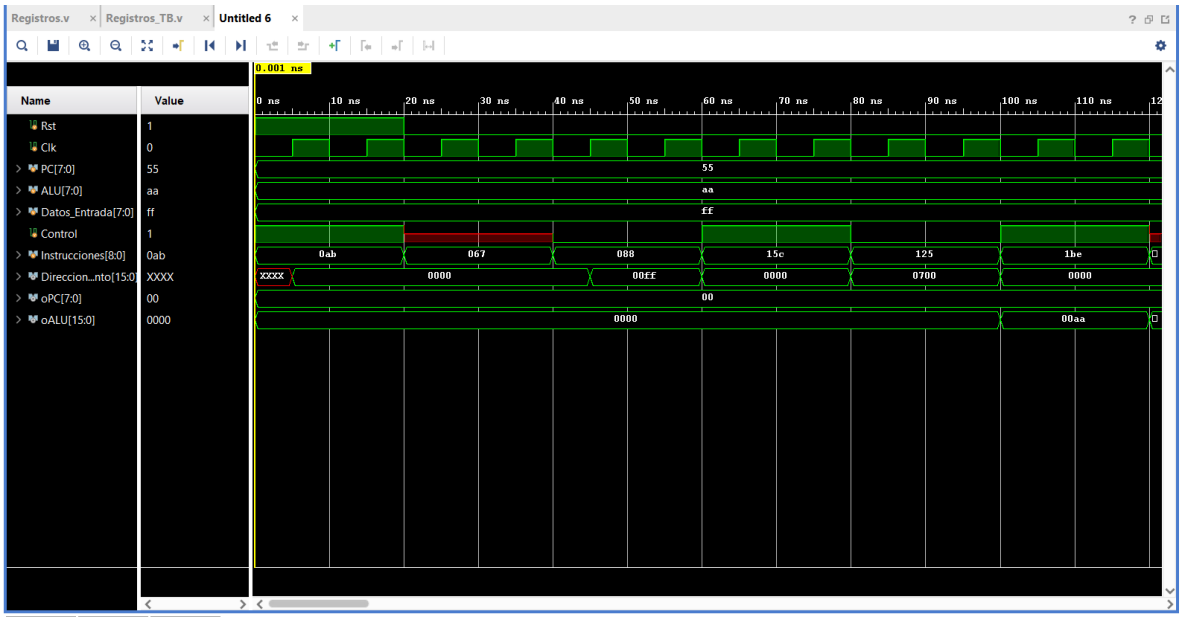

Figura 20 Simulación Registros

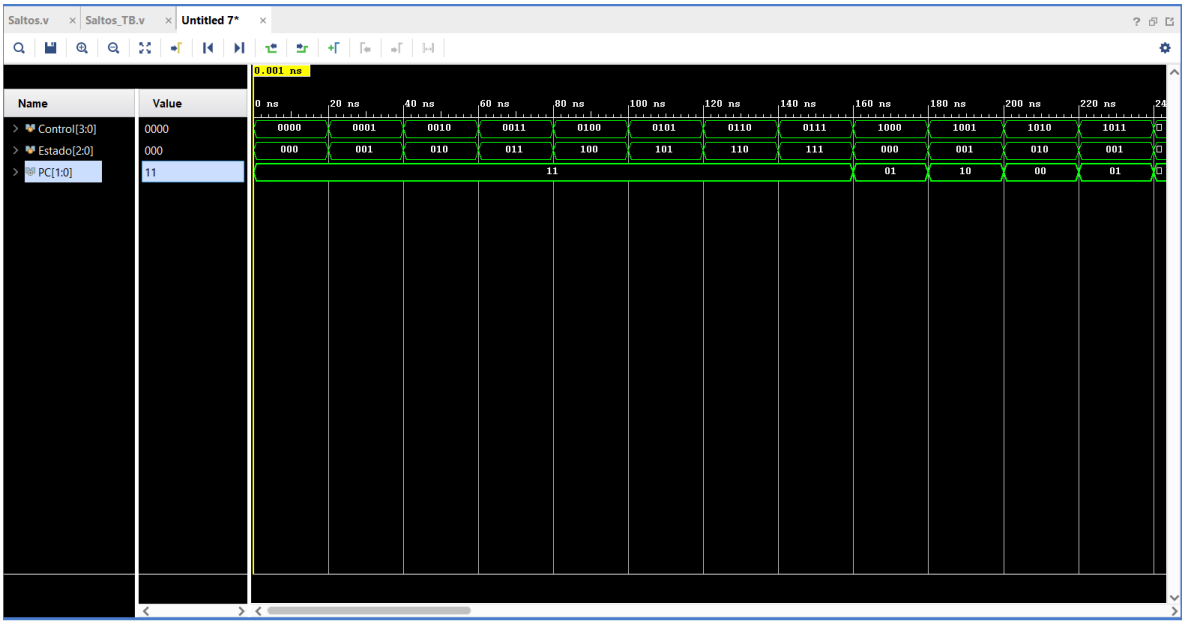

Figura 21 Simulación Saltos

Diagrama síntesis

En la ilustración 22 presentamos el esquemático del micro controlador, puede apreciarse claramente su estructura tipo Harvard.

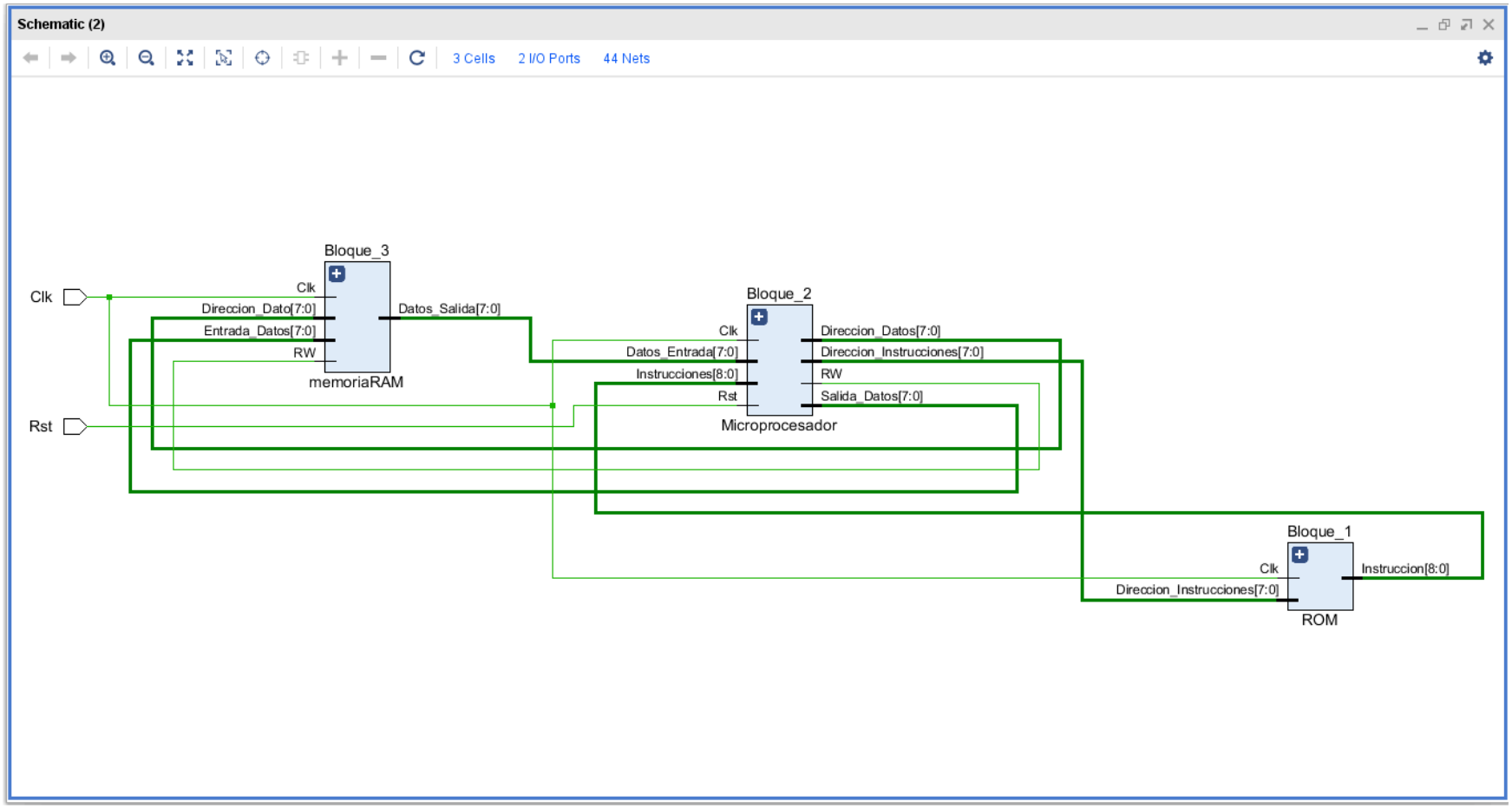

Figura 22 Esquemático Micro controlador

A continuación, en la ilustración 23 se muestra el esquemático generado por vivado del microprocesador sintetizado.

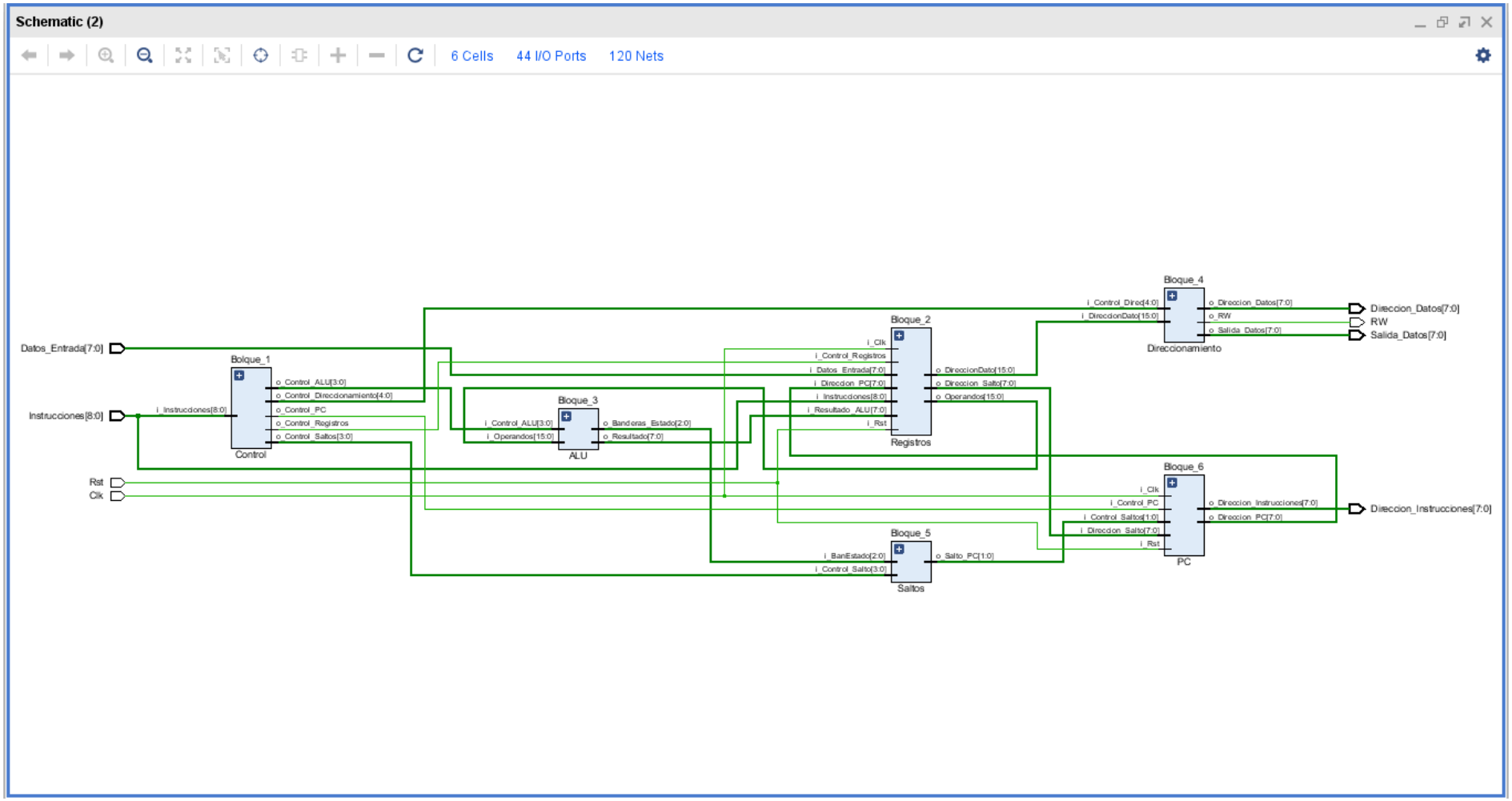

Figura 23 Esquemático Microprocesador

Project summary

La sección *settings (Figura24)* especifica las configuraciones básicas que tendrá el diseño de hardware, como se muestra en la *ilustración* X se encuentra el nombre del proyecto, la ruta en la que se encuentra almacenada, la familia del producto sobre la cual se implementa el diseño, el modelo del FPGA (en este caso la basys 3), el nombre del módulo principal, el lenguaje sobre el cual está basado y el lenguaje en el que se simula.

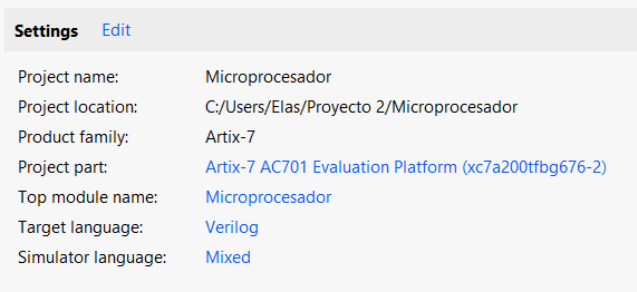

Figura 24 Settings

La sección *síntesis (ilustración 25)* se encuentra la información referente al proceso de sintetizado, dando a conocer algún error o advertencia durante el proceso.

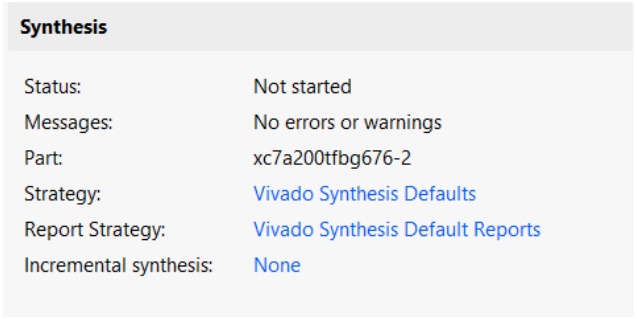

Figura 25 Synthesis

En la sección *implementación (ilustración 26)* se encuentra lo referente a esto. Se encuentra el estado y los errores y advertencias.

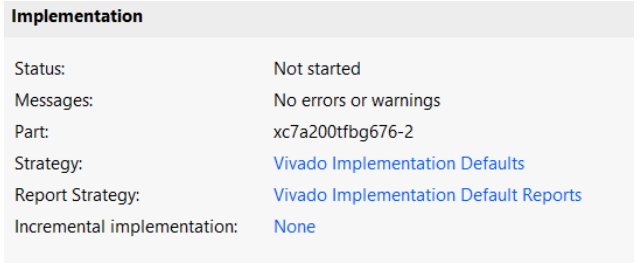

Figura 26 Implementation

La sección *DRC violations (ilustración 27),* se refiere a las reglas de diseño, es decir, indica si hay alguna violación en las reglas o algo fuera del estándar que no genera un error como tal.

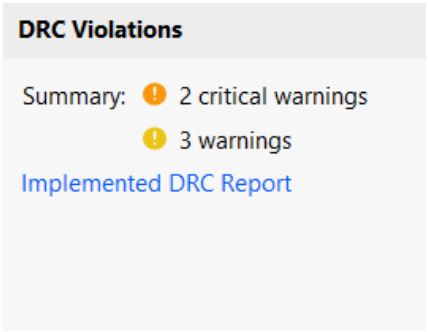

Figura 27 DRC violations

La sección *utilización (ilustración 28)* se refiere a la cantidad de recursos utilizados por parte de la tarjeta.

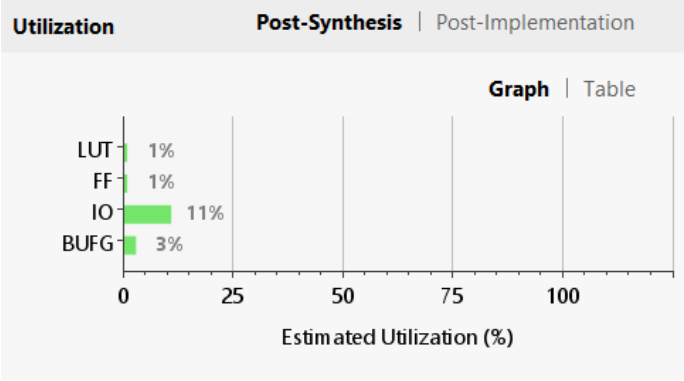

Figura 28 Utilization

La sección *power (ilustración 29)* se refiere a la información útil sobre el consumo energético de la tarjeta, en otras palabras, qué tanta potencia consumirá y qué tanto incrementará su temperatura al momento de implementar el diseño sobre el FPGA.

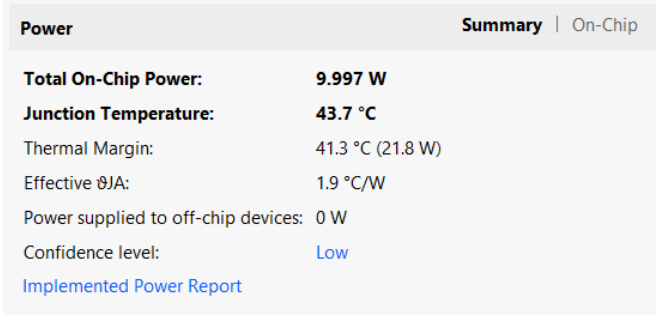

Figura 29 Power

Análisis de resultados

Se realizó una descripción de un microprocesador de 8 bits con el propósito de conocer tipos de microprocesadores, su estructura básica y especificaciones, además con el uso de un lenguaje de descripción de hardware logramos concluir este proyecto, ya que éste nos ayuda a describir y simular nuestro procesador.

Para realizar este proyecto es necesario seguir una metodología, por lo que para iniciar debemos tener el diseño de caja negra y comprenderlo, es decir conocer la función que realiza cada una de sus entradas y salidas, además de conocer claramente que operaciones puede realizar este procesador así como sus limitantes, comprender esto requiere un conocimiento previo de teoría sobre los procesadores.

En dicha teoría necesitaremos reafirmar conocimientos específicos para la realización de este microprocesador, ya que deberá ser diseñado bajo la arquitectura Harvard, por ello es de suma importancia conocer las partes que componen a esta arquitectura, así como sus ventajas y desventajas frente a otras arquitecturas.

Una vez comprendido el diagrama de caja negra, así como sus posibles operaciones y sus limitantes debemos comenzar a realizar el diagrama de caja blanca. Como aprendimos en la teoría cada microprocesador tiene características particulares como lo es su arquitectura, limitaciones y atributos, también de partes básicas, que le permiten realizar las operaciones para el cual fue diseñado, como es el caso de la ALU (Unidad Aritmética Lógica), que nos ayuda a realizar las operaciones entre los registros y entre operadores para posteriormente almacenarse dicha operación como un registro.

También otro bloque esencial entre los microprocesadores es el de los registros, este bloque es muy importante ya que en él se almacenarán operaciones que se deseen guardar, así como registros con los cuales realizar operaciones, así como almacenar un conjunto de operaciones cuando se vaya a realizar un salto.

El bloque de control recibe la instrucción y es el encargado de distribuirla, es decir en la instrucción contiene 3 bits dedicados a la operación a realizar, otros 3 bits para el primer operador y los últimos 3 bits para el segundo operador, y es el encargado de controlar lo que la ALU en caso de que la operación a realizar sea dominio de este bloque, si la operación no es del dominio de la ALU controla los bloques a los cuales la instrucción pertenece. Este bloque se encarga del manejo del microprocesador siendo una parte muy importante.

En el bloque de saltos es donde se interrumpen las operaciones actuales para darle paso a una con mayor importancia por ende las instrucciones interrumpidas son guardadas en registro para no perder la continuidad después del salto.

PC es el encargado de contener la dirección de la siguiente instrucción a ejecutar

El bloque direccionamiento se encarga de enviar las direcciones de los datos, así como el bus de datos a la memoria de datos.

Una vez que sabemos los bloques esenciales que llevará nuestro microprocesador debemos realizar el diagrama de caja blanca, con sus conexiones, nombrando también la señal interna que une un bloque con otro, así como su dimensión, es decir el tamaño de bus requerido para esa señal.

Una vez hecho el diagrama de caja blanca procedemos a hacer un pseudocódigo para resolver cómo será el funcionamiento de cada bloque, este seudocódigo debe presentar las características básicas de un lenguaje de descripción de hardware, para mejor entendimiento.

Teniendo todos los pasos anteriores cumplidos procedemos a realizar la codificación en vivado, herramienta que nos ayudará a simular nuestro microprocesador. Al pasar de pseudocódigo a código en verilog verificamos que la lógica con la cual nuestro bloque sea la misma con la cual trabaja nuestra herramienta para evitar errores que al final resultarán en pérdida de tiempo.

Realizamos pruebas corroborando su funcionamiento tanto en sus operaciones, así como las limitaciones que teóricamente debe de tener.

Finalmente implementamos la arquitectura Harvard, lo visto en la teoría se lleva a la práctica creando un programa de una memoria RAM (memoria de acceso aleatorio), esta memoria es un almacenamiento donde se guardan los datos procesados o a procesar por el microprocesador, también creamos un programa que servirá como memoria ROM (memoria de solo lectura), es decir esta memoria mandará las instrucciones a realizar, y dichas instrucciones estarán almacenadas en el interior de este bloque mediante un archivo de extensión .mem sin posibilidad de ser modificadas. Y por último creamos el programa TOP que unirá el microprocesador, la memoria RAM y ROM en un mismo sistema capaz de realizar las operaciones establecidas sin errores y con resultados correctos.

Conclusiones

Finalmente, se obtuvo un microprocesador de 8 bits con arquitectura Harvard capaz de realizar operaciones matemáticas y lógicas a un conjunto de datos. Aparentemente, las simulaciones funcionan de forma correcta, por lo tanto, se espera que, al conectarle las memorias externas, tenga un buen funcionamiento. Se propuso realizar una multiplicación con los recursos con los que cuenta el diseño realizado, por ello, se hicieron múltiples intentos, teniendo el resultado final el cual se mostrará en la presentación del presente.

Es muy importante tanto para el diseño como para el desarrollo de un proyecto o sistema que busque cumplir un requisito establecido o una funcionalidad específica el realizarlo de forma ordenada y siguiendo las jerarquías de diseño, en nuestro caso de un sistema digital, el seguimiento del flujo de diseño establecido de forma correcta y prestando la justa atención a cada parte ayuda a evitar posibles complicaciones futuras durante los pasos posteriores hasta completar el desarrollo de todo el sistema, no sólo para evitar complicaciones en algún punto o que si llegase a ocurrir poder identificarlo con facilidad y emplear menos tiempo en resolverlo, sino que además mejora y facilita el trabajo en equipo ayudando a evitar complicaciones tediosas a otros grupos de trabajo para conseguir el fin deseado en tiempo y sin mayores problemáticas esto es algo que también nos ha dejado como aprendizaje en la realización de este proyecto.

Lo más complicado del proceso de elaboración del microprocesador, fue sin duda alguna entender los conceptos clave que trae consigo la metodología que se describe en el presente documento, esto, debido a que se trata de un gran número de conceptos nuevos, y en ocasiones confusos, sin embargo, se concluye que se tuvo un gran enriquecimiento de la materia, que seguro será útil en un futuro profesional. Otra dificultad, fue la realización de las memorias, debido a que se tuvo que implementar esta idea desde cero en poco tiempo.

Finalmente, se tuvo un buen resultado, solo queda consultarlo con el docente y verificar que, efectivamente, funciona de manera correcta.

Referencias

[1] J. A. SAINZ, INTRODUCCIÓN A LOSMICROPROCESADORES, E.U.I.T.I. VITORIA-GASTEIZ: CATEDRÁTICO E.U., 2000.

[2] Anónimo, «Significados,» 15 02 2019. [En línea]. Available: https://www.significados.com/microprocesador/#:~:text=La%20Unidad%20Central%20de%20Procesos,hasta%20con%20millones%20de%20transistores.. [Último acceso: 22 11 2020].

[3] R. Camacho, «Computo Integrado,» 2012 04 9. [En línea]. Available: http://rcmcomputointegrado.blogspot.com/2012/04/arquitectura-von-neumann.html. [Último acceso: 22 11 2020].

Apéndice A (Códigos)

Microcontrolador.v

```
module Microcontrolador(
input Clk,
input Rst
);
wire [7:0] DatosM_RAM;
wire [7:0] DatosRAM_M;
wire [7:0] DireccionDatos;
wire RW;
wire [7:0] DireccionInstrucciones;
wire [8:0] BusInstrucciones;

ROM Bloque_1 (
.Clk (Clk),
.Direccion_Instrucciones (DireccionInstrucciones),
.Instruccion (BusInstrucciones)
);

Microprocesador Bloque_2 (
.Clk (Clk),
.Rst (Rst),
.Datos_Entrada (DatosRAM_M),
.Instrucciones (BusInstrucciones),
.Direccion_Instrucciones (DireccionInstrucciones),
.Direccion_Datos (DireccionDatos),
.Salida_Datos (DatosM_RAM),
.RW (RW)
);

memoriaRAM Bloque_3 (
.Clk (Clk),
.Direccion_Dato (DireccionDatos),
.Entrada_Datos (DatosM_RAM),
.RW (RW),
.Datos_Salida (DatosRAM_M)
);

endmodule
```

memoriaRAM.v

```
module memoriaRAM(
input Clk,
input [7:0] Direccion_Dato,
input [7:0] Entrada_Datos,
input RW,
output [7:0] Datos_Salida
);


parameter RAMFILE = "Datos.mem";
reg [7:0] RAM [0:255];
wire [7:0] EntradaDatos;

assign EntradaDatos = Entrada_Datos;

always@(posedge Clk)
begin
if(RW)
RAM[Direccion_Dato] <= Entrada_Datos; // escritura
end

assign Datos_Salida = (RW == 1'b0) ? RAM[Direccion_Dato]: Datos_Salida;
//lectura


//Se inicializa la memoria RAM
initial
begin
$readmemh(RAMFILE, RAM);
end
endmodule
```

ROM.v

```
module ROM(
input Clk,
input [7:0] Direccion_Instrucciones,
output [8:0] Instruccion
);

parameter ROMFILE = "Instrucciones.mem";
reg [8:0] ROM [0:255];
reg [8:0] Instrucciones;

assign Instruccion = Instrucciones;

always@(negedge Clk)
begin
Instrucciones <= ROM[Direccion_Instrucciones];
end

//Se inicializa la memoria ROM
initial
begin
$readmemb(ROMFILE, ROM);
/* ROM[0] = 9'h0;
ROM[1] = 9'h1;
ROM[2] = 9'h2;
ROM[3] = 9'h3;
ROM[4] = 9'h4;
ROM[5] = 9'h5;
ROM[6] = 9'h6;
ROM[7] = 9'h7; */
end

endmodule
```

Microprocesador.v

```
module Microprocesador(
//Declara las entradas y salidas del microprocesador
    input Clk,      //Entrada señal cuadrada con frecuencia de 100Mhz
    input Rst,      //Entrada que reinicia valores a 0
    input [7:0] Datos_Entrada,
//Entrada de datos que serán procesados guardados temporalmente
    input [8:0] Instrucciones,
//Entrada que contiene la instrucción a realizar
    output [7:0] Direccion_Instrucciones,
//Salida que controla la dirección de memoria de las instrucciones
    output [7:0] Direccion_Datos,
//Salida que controla las direcciones de la memoria de datos
    output [7:0] Salida_Datos,
//Salida de los datos procesados que son almacenados en la memoria
    output RW
//Salida que indica la acción de la memoria, si es lectura o escritura
    );

  wire Control_PC;      //Declaración de las conexiones que unirán los
//diferentes bloques del microprocesador
  wire Control_L;
  wire [3:0] Control_ALU;
  wire [4:0] Control_D;
  wire [3:0] Control_Salto;
  wire [15:0] Operandos;
  wire [7:0] Resultado;
  wire [7:0] Direccion;
  wire [15:0] Direccion_Dato;
  wire [2:0] Banderas;
  wire [1:0] Salto_PC;
  wire [7:0] Direccion_PC;

  Control Bolque_1 (        //Conexiones del bloque Control, se conectan
//las entradas y salidas con señales internas o salidas y entradas
    .i_Instrucciones (Instrucciones),
    .o_Control_PC (Control_PC),
    .o_Control_Registros (Control_L),
    .o_Control_ALU (Control_ALU),
    .o_Control_Direccionamiento (Control_D),
    .o_Control_Saltos (Control_Salto)
  );

  Registros Bloque_2 (     //Conexiones del bloque Registros, se conectan
//las entradas y salidas con señales internas o salidas y entradas
    .i_Rst (Rst),
    .i_Clk (Clk),
    .i_Direccion_PC (Direccion_PC),
    .i_Resultado_ALU (Resultado),
    .i_Control_Registros (Control_L),
```

```
    .i_Datos_Entrada (Datos_Entrada),
    .i_Instrucciones (Instrucciones),
    .o_Direccion_Salto (Direccion),
    .o_DireccionDato (Direccion_Dato),
    .o_Operandos (Operandos)
  );

  ALU Bloque_3 (        //Conexiones del bloque ALU, se conectan las
//entradas y salidas con señales internas o salidas y entradas
    .i_Control_ALU (Control_ALU),
    .i_Operandos (Operandos),
    .o_Resultado (Resultado),
    .o_Banderas_Estado (Banderas)
  );

  Direccionamiento Bloque_4 (   //Conexiones del bloque Direccionamiento,
//se conectan las entradas y salidas con señales internas o salidas y
//entradas
    .i_Control_Direc (Control_D),
    .i_DireccionDato (Direccion_Dato),
    .o_Direccion_Datos (Direccion_Datos),
    .o_Salida_Datos (Salida_Datos),
    .o_RW (RW)
  );

  Saltos Bloque_5 (     //Conexiones del bloque Saltos, se conectan las
//entradas y salidas con señales internas o salidas y entradas
    .i_Control_Salto (Control_Salto),
    .i_BanEstado (Banderas),
    .o_Salto_PC (Salto_PC)
  );

  PC Bloque_6 (         //Conexiones del bloque PC, se conectan las
//entradas y salidas con señales internas o salidas y entradas
    .i_Clk (Clk),
    .i_Rst (Rst),
    .i_Control_PC (Control_PC),
    .i_Control_Saltos (Salto_PC),
    .i_Direccion_Salto (Direccion),
    .o_Direccion_Instrucciones (Direccion_Instrucciones),
    .o_Direccion_PC (Direccion_PC)
  );

endmodule       //Finaliza el bloque Microprocesador
```

ALU.v

```
module ALU(
// Se declaran las entradas y salidas del bloque ALU
    input [3:0] i_Control_ALU,
// Entrada proveniente del bloque control
    input [15:0] i_Operandos,
//Entrada que carrea los operandos desde el bloque registros
    output [7:0] o_Resultado,
//Salida que arroja el resultado de la operacion aritmética lógica
    output [2:0] o_Banderas_Estado
// Salida que indica el estado de las banderas
    );

    reg signed [8:0] Resultado = 0;
//Registro que guarda temporalemente el valor del resultado
    wire signed [7:0]op1;
//Cables que conectan las partes del bloque
    wire signed [7:0]op2;
    wire [2:0] Operacion;
    wire Control;
    wire Z;
    wire N;
    wire C;

    assign Control = i_Control_ALU[3];
// En esta sección se asinan los valores de un elemento a otro
    assign Operacion = i_Control_ALU[2:0];
    assign op1 = i_Operandos[7:0];
    assign op2 = i_Operandos[15:8];

always @(i_Control_ALU or Operacion or op1 or op2)
//Bloque que indica el proceso donde se hacen las operaciones
begin
    if (Control)    //Si existe señal desde control
    begin
        case (Operacion)
            3'b000: Resultado <= op1 + op2;  //Suma de operandos
            3'b001: Resultado <= op1 - op2;  //Resta de operandos
            3'b010: Resultado <= op1 << op2; //Corrimiento a la izquierda
            3'b011: Resultado <= op1 >> op2; //Corrimiento a la derecha
            3'b100: Resultado <= ~op2;       //Negación del operando
            3'b101: Resultado <= op1 & op2;  //Operación lógica and
            3'b110: Resultado <= op1 | op2;  //Operación lógica or
            default: Resultado <= op1 ^ op2; //Operacion lógica xor
        endcase
    end
    else
        Resultado <= Resultado;
end

assign Z = (Resultado == 0) ? 1'b1 : 1'b0;
//Si el resutlado es 0 la bandera Z toma valor de 1
```

```
assign N = (Resultado[8] == 1) ? 1'b1 : 1'b0;
//Si el bit 8 es 1 el resultado corresponde a un valor negativo
assign C = (Resultado[8] == Resultado[7]) ? 1'b0 : 1'b1;
//Si el resultado del bit 8 es igual al del bit 7 se activa flag carry
assign o_Resultado = Resultado[7:0];        //Resultado de 8 bits
assign o_Banderas_Estado = {Z, N, C};
//Asingna la salida de las banderas

endmodule       //Finaliza el bloque
```

Control

```
module Control(
    input [8:0] i_Instrucciones,
//Entrada en la que se encuentran los argumentos del set de instrucciones
    output o_Control_PC,
//Salida al PC para indicar el seguimiento de las instrucciones
    output o_Control_Registros,
//Salida que manda datos volátiles al bloque registros
    output [3:0] o_Control_ALU,
//Salida que indica si se se requiere realizar una operación en el bloque
//ALU y la operación
    output [4:0] o_Control_Direccionamiento,
//Salida que lleva el código de la instrucción
    output [3:0] o_Control_Saltos
//Salida que controla la ejecución de los saltos
    );

    wire [2:0] c_in;
//Declaración de conexiones para llevar a cabo el proceso
    wire [2:0] c_reg;
    wire [2:0] c_arg;
    wire PC;
    wire Registros;
    wire [3:0] ALU;
    wire [4:0] Direccionamiento;
    wire [3:0] Saltos;

    assign c_in = i_Instrucciones [8:6];
//Asignación de los valores que toman las conexiones respecto a entradas
    assign c_reg = i_Instrucciones [5:3];
    assign c_arg = i_Instrucciones [2:0];
    assign o_Control_PC = PC;
//Asignación del valor de las salidas con respecto a los datos manejados
en el bloque
    assign o_Control_Registros = Registros;
    assign o_Control_ALU = ALU;
    assign o_Control_Direccionamiento = Direccionamiento;
    assign o_Control_Saltos = Saltos;

 assign Direccionamiento = (c_in == 3'b010) ? {2'b01, c_arg} :
//Determina el estado de la salida de direccionamiento
                           (c_in == 3'b011) ? {2'b10, c_arg} :
                           (c_in == 3'b100) ? {2'b11, c_arg} : 5'b0;

 assign PC = (c_in == 3'b111) ? 1'b0 : 1'b1;
//Asigna valor a PC dependienod de c_in

 assign Registros = (c_in == 3'b110) ? 1'b1 : 1'b0;

 assign ALU = (c_in == 3'b110) ? {1'b1,c_arg} :
```

```
                (c_in == 3'b000) ? 4'b0 : {1'b0,c_arg};

 assign Saltos = (c_in == 3'b111) ? {1'b1,c_arg} :
                 (c_in == 3'b000) ? 4'b0 : {1'b0,c_arg};


endmodule        //Finaliza el bloque
```

PC.v

```
module PC(
    input i_Clk,        //Entrada de reloj de 100MHZ
    input i_Rst,
//Entrada de reseteo, los valores toman valores de 0
    input i_Control_PC,
//Entrada que va del bloque control al PC para direccionar las
instrucciones
    input [1:0] i_Control_Saltos,
//Entrada que indica si la instruccion se salta
    input [7:0] i_Direccion_Salto,
//Entrada que indic la dirección del salto
    output [7:0] o_Direccion_Instrucciones,
//Salida al exterior del micro que indica la dirección de las
instrucciones
    output [7:0] o_Direccion_PC         //Salida dirigida a los registros
    );

    wire Control;
//Declaración de cables y registros para llevar a cabo el funcionamiento
    wire [1:0] Saltos;
    wire [7:0] Direccion_Salto;
    wire [7:0] Direccion_PC;
    reg [7:0] contador = 0;

assign Control = i_Control_PC;
//Asigna los valores a los registros y salidas
assign Saltos = i_Control_Saltos;
assign Direccion_Salto = i_Direccion_Salto;
//assign o_Direccion_Instrucciones = Direccion_Instrucciones;
assign  o_Direccion_PC = Direccion_PC;

always @(posedge i_Clk)     //Determina el proceso que se llevará a cabo
begin
     if(i_Rst) begin
//Si la señal de reset es 1, el contador se reinicia
        contador <= 0;
     end
     else begin
         if(Control) begin
            contador <= contador +1;
//Incrementa el contador cada que se mande señal de control
            if (contador == 255)
//Reinicia el contador cuando se llegue al límite
                contador <= 0;
         end
         else begin
            case (Saltos)
                2'b00:begin
// Suma uno más a la cuenta porque se cedió el salto
```

```
                    contador <= contador +1;
                    if (contador == 255)
                        contador <= 0;
                end
                2'b01:begin
//El contador toma el valor de la direccion del salto
                    contador <= Direccion_Salto;
                end
                2'b10:begin
//El contador suma un numero más a la cuenta
                    contador <= contador +1;
                    if (contador == 255)
                        contador <= 0;
                end

            endcase

        end

    end

end

 assign o_Direccion_Instrucciones = contador;
//Asigna el valor del contador a la salida de dirección de instrucciones
 assign Direccion_PC = (Saltos == 2'b10) ? o_Direccion_Instrucciones :
8'b0;        //Selecciona el valor de la dirección de direccion_PC
dependiendo del valor del estado de saltos
endmodule       //Termina el bloque
```

Registros.v

```
module Registros(
    input i_Rst,        //Entrada de reloj de 100MHZ
    input i_Clk,
//Entrada de reseteo, los valores toman valores de 0
    input [7:0] i_Direccion_PC,
//Entrada que proviene de PC que trae el estado del bloque guardada en R7
    input [7:0] i_Resultado_ALU,        //Entrada que proviene de ALU que
lleva el valor del resultado se guarda en R0
    input i_Control_Registros,
//Entrada que controla la carga de valores
    input [7:0] i_Datos_Entrada,
//Entrada que contiene los datos que provienen de la memoria
    input [8:0] i_Instrucciones,
//Entrada que contiene el códogo de la instrucción a ejecutar
    output  [7:0] o_Direccion_Salto,
//Salida que indica a PC si se va a saltar de instrucción
    output [15:0] o_DireccionDato,      //Salida que contiene el dato que
se almacena en la memoria para realizar la carga
    output  [15:0] o_Operandos          //Salida que va a la ALU que
contiene los operandos de los que se va a hacer la operación
 );

reg[7:0] BancoRegistros [0:7];
//Se declara un registro para almacenar datos temporalmente
integer i;
wire [2:0] c_Ins;       //Se declaran conexiones
wire [2:0] c_RX;
wire [2:0] c_RY;

assign c_Ins = i_Instrucciones[8:6];
//Se asigna a cada conexion a un segmentento del bus de instrucciones
assign c_RX = i_Instrucciones[5:3];
assign c_RY = i_Instrucciones[2:0];

always@(posedge i_Clk)
//Cada que el reloj tiene un flanco positivo se ejecuta el proceso
begin
      if (i_Rst)
//Cuando Rst es 1, el contenido de registros se hace cero
        for(i = 0; i <8; i = i+1)
            BancoRegistros[i] <= 0;

      else begin
        case(c_Ins)     //Dependiendo de la instrucción, el banco de
registros reacciona de distinta forma
          3'b001: BancoRegistros [c_RX] <= {5'b00000, c_RY};
          3'b010: BancoRegistros[c_RX] <= i_Datos_Entrada;
          3'b101: BancoRegistros [c_RX] <= BancoRegistros[c_RY];
          3'b111: begin
```

```
                    if (c_RY == 3'b001)
                        BancoRegistros[7] <= i_Direccion_PC;
                end
        endcase

        if (i_Control_Registros)
//Guarda los datos del resultado en banco de registros
            BancoRegistros[000] <= i_Resultado_ALU;
       end

end

assign o_Direccion_Salto = (c_Ins == 3'b111) ? BancoRegistros[c_RX] :
8'b0;     // La direccion de salto toma el valor del banco de registros o
ceros dependiendo de c_Ins
assign o_Operandos = (c_Ins == 3'b110) ? {BancoRegistros[c_RX],
BancoRegistros[000]} : 16'b0;

assign o_DireccionDato = (c_Ins == 3'b010) ? { BancoRegistros[c_RY],
BancoRegistros[c_RX] } :
//Asigna valor de la salida del direccionamiento del dato
                         (c_Ins == 3'b011) ? { BancoRegistros[c_RX],
BancoRegistros[c_RY] } :
                         (c_Ins == 3'b100) ? { BancoRegistros[c_RX],
BancoRegistros[c_RY] } : 16'b0;
endmodule       //Finaliza el bloque Registros
```

Saltos.v

```
module Saltos(      //Declara las entradas y salidas del bloque
    input [3:0] i_Control_Salto,        //Entrada que carrea el código de
condición a verificar
    input [2:0] i_BanEstado,            //Entrada para indicar la
presencia de banderas
    output [1:0] o_Salto_PC             //Salida que indica a PC que se
va a realizar un salto
    );

    wire [3:0] Control;     //Se declaran las conexiones para el
movimiento de datos
    wire [2:0] Estado;
    reg [1:0] PC;       //Registro que envia estado al PC

    assign o_Salto_PC = PC;     //Asigna los valores de salida respecto
al registro
    assign Control = i_Control_Salto;       //Las conexiones toman el
valor de entradas
    assign Estado = i_BanEstado;

always @(i_Control_Salto or i_BanEstado)        //Cada que ocurra un
salto desde el bloque control o el estado de banderas cambie
begin

    if(Control[3]) begin
        case (Control[2:0])         //Case para definir el estado que
tomará la salida hacia PC
            3'b001:begin
                PC <= 2'b10;
            end
            3'b010:begin
                if(Estado[2]) begin
                    PC <= 2'b01;
                end
                else begin
                    PC <= 2'b00;
                end
            end

            3'b011:begin
                if(!Estado[2]) begin
                    PC <= 2'b01;
                end
                else begin
                    PC <= 2'b00;
                end
            end

            3'b100:begin
```

```
                if(Estado[1]) begin
                    PC <= 2'b01;
                end
                else begin
                    PC <= 2'b00;
                end
            end

            3'b101:begin
                if(!Estado[1]) begin
                    PC <= 2'b01;
                end
                else begin
                    PC <= 2'b00;
                end
            end

            3'b110:begin
                if(Estado[0]) begin
                    PC <= 2'b01;
                end
                else begin
                    PC <= 2'b00;
                end
            end

            3'b111:begin
                if(!Estado[1]) begin
                    PC <= 2'b01;
                end
                else begin
                    PC <= 2'b00;
                end
            end

            default: begin
                PC <= 2'b01;
            end
        endcase
    end

    else begin
        PC <= 2'b11;
    end

end

endmodule       //Finaliza el bloque
```

Saltos.v

```
module Saltos(      //Declara las entradas y salidas del bloque
    input [3:0] i_Control_Salto,
//Entrada que carrea el código de condición a verificar
    input [2:0] i_BanEstado,
//Entrada para indicar la presencia de banderas
    output [1:0] o_Salto_PC
//Salida que indica a PC que se va a realizar un salto
    );

    wire [3:0] Control;
//Se declaran las conexiones para el movimiento de datos
    wire [2:0] Estado;
    reg [1:0] PC;       //Registro que envia estado al PC

    assign o_Salto_PC = PC;
//Asigna los valores de salida respecto al registro
    assign Control = i_Control_Salto;
//Las conexiones toman el valor de entradas
    assign Estado = i_BanEstado;

always @(i_Control_Salto or i_BanEstado)        //Cada que ocurra un
salto desde el bloque control o el estado de banderas cambie
begin

    if(Control[3]) begin
        case (Control[2:0])
//Case para definir el estado que tomará la salida hacia PC
            3'b001:begin
                PC <= 2'b10;
            end
            3'b010:begin
                if(Estado[2]) begin
                    PC <= 2'b01;
                end
                else begin
                    PC <= 2'b00;
                end
            end

            3'b011:begin
                if(!Estado[2]) begin
                    PC <= 2'b01;
                end
                else begin
                    PC <= 2'b00;
                end
            end
```

```
            3'b100:begin
                if(Estado[1]) begin
                    PC <= 2'b01;
                end
                else begin
                    PC <= 2'b00;
                end
            end

            3'b101:begin
                if(!Estado[1]) begin
                    PC <= 2'b01;
                end
                else begin
                    PC <= 2'b00;
                end
            end

            3'b110:begin
                if(Estado[0]) begin
                    PC <= 2'b01;
                end
                else begin
                    PC <= 2'b00;
                end
            end

            3'b111:begin
                if(!Estado[1]) begin
                    PC <= 2'b01;
                end
                else begin
                    PC <= 2'b00;
                end
            end

            default: begin
                PC <= 2'b01;
            end
        endcase
    end

    else begin
        PC <= 2'b11;
    end

end

endmodule       //Finaliza el bloque
```

Direccionamiento.v

```
module Direccionamiento(
//Se declaran las entradas y salidas del bloque
    input [4:0] i_Control_Direc,
//Entrada que trae la instrucción que se ejecutará
    input [15:0] i_DireccionDato,       //Entrada que proviene de
registros donde viene la dirección de donde se guardará el dato
    output [7:0] o_Direccion_Datos,     //Salida que va a la memoria en
la que se define la dirección
    output [7:0] o_Salida_Datos,
//Va a la memoria y lleva el dato a almacenar
    output reg o_RW         //Define si se trata de lectura o escritura
 );

 wire [1:0] c_in;       //Declara conexiones entre bloques
 wire [2:0] c_dat;
 wire [7:0] Datos;
 reg [7:0] Salida_Datos = 0;
 reg [7:0] Direccion_Datos = 0;

 assign c_in = i_Control_Direc[4:3];
//Segmenta la entrada de i_Control_Direc
 assign c_dat = i_Control_Direc[2:0];
 assign o_Salida_Datos = Salida_Datos;
//Asigna los datos de los registros a las salidas
 assign o_Direccion_Datos = Direccion_Datos;

assign Datos = (c_dat == 3'b000) ? 8'b00000001:         //1
               (c_dat == 3'b001) ? 8'b00000010:         //2
               (c_dat == 3'b010) ? 8'b00000100:         //4
               (c_dat == 3'b011) ? 8'b00001000:         //8
               (c_dat == 3'b100) ? 8'b00010000:         //16
               (c_dat == 3'b101) ? 8'b00100000:         //32
               (c_dat == 3'b110) ? 8'b01000000:         //64
               (c_dat == 3'b111) ? 8'b10000000: 8'h00;  //128 :00

always@(c_in)
begin
    if (c_in == 2'b00)
        Direccion_Datos <= Direccion_Datos;
//La salida de datos toma el valor de direccionamiento
    else
        Direccion_Datos <= i_DireccionDato[15:8];
//La salida de datos toma el valor de direccionamiento

    case(c_in)
    2'b01: begin
            Salida_Datos <= i_DireccionDato[7:0];
//La salida de datos toma el valor de direccionamiento
```

```
            o_RW <= 0;      //La salida RW indica que es lectura
          end
    2'b10: begin
            Salida_Datos <= Datos;
            o_RW <= 1;          //La salida RW indica que es escritura
          end
    2'b11: begin
            Salida_Datos <= i_DireccionDato[7:0];
//La salida de datos toma el valor de direccionamiento
            o_RW <= 1;
//La salida RW indica que es escritura
          end
    default: begin
              Salida_Datos <= Salida_Datos;
              o_RW <= 0;        //La salida RW indica que es lectura
             end
    endcase

end
endmodule
```

Apéndice B (Test Bench)

Microcontrolador_TB.v

```
module Microcontrolador_TB(

);

reg Clk;
reg Rst;

Microcontrolador DUT (
.Clk (Clk),
.Rst (Rst)
);

initial
begin
Clk <= 0;
Rst <= 1;
#10 Rst <= 0;

end

always@ (Clk)
#5 Clk <= ~Clk;

endmodule
```

memoriaRAM_TB.v

```
module memoriaRAM_TB(
);

parameter RAMFILE = "Datos.mem";
reg Clk;
reg [7:0] Direccion_Dato;
reg [7:0] Entrada_Datos;
reg RW;
wire [7:0] Datos_Salida;

memoriaRAM #(.RAMFILE(RAMFILE)) DUT(
.Clk (Clk),
.Direccion_Dato (Direccion_Dato),
.Entrada_Datos (Entrada_Datos),
.RW (RW),
.Datos_Salida (Datos_Salida)
);
initial
begin
Clk <= 0;
Direccion_Dato <= 8'b00000001; RW = 0;
Entrada_Datos <= 8'b11111111;
#20;
Direccion_Dato <= 8'b00000010; RW = 0;
Entrada_Datos <= 8'b11100010;
#20;
Direccion_Dato <= 8'b00000011; RW = 1;
Entrada_Datos <= 8'b11100011;
#20;
Direccion_Dato <= 8'b00000011; RW = 0;
Entrada_Datos <= 8'b000000011;
#20;
Direccion_Dato <= 8'b00000101; RW = 1;
Entrada_Datos <= 8'b11111000;
#20;
Direccion_Dato <= 8'b00000110; RW = 0;
Entrada_Datos <= 8'b11110000;
#20;
end
initial begin
//-- Fichero donde almacenar los resultados
$dumpfile("memoriaRAM_TB.vcd");
$dumpvars(0, memoriaRAM_TB);
// # 100 $display("FIN de la simulacion");
// $finish;
end
always@(Clk)
#5 Clk <= ~Clk;
endmodule
```

memooriaROM_TB.v

```
module memooriaROM_TB(
);

parameter ROMFILE = "Instrucciones.mem";
reg Clk;
reg [7:0] Direccion_Instrucciones;
wire [8:0] Instruccion;

ROM #(.ROMFILE(ROMFILE)) DUT(
.Clk (Clk),
.Direccion_Instrucciones (Direccion_Instrucciones),
.Instruccion (Instruccion)
);

// always #5 Clk <= ~Clk;

initial
begin
Clk <= 0;
Direccion_Instrucciones <= 8'b00000001;
#20;
Direccion_Instrucciones <= 8'b00000010;
#20;
Direccion_Instrucciones <= 8'b00000011;
#20;
Direccion_Instrucciones <= 8'b00000100;
#20;
Direccion_Instrucciones <= 8'b00000101;
#20;
Direccion_Instrucciones <= 8'b00000110;
#20;
end
initial begin
//-- Fichero donde almacenar los resultados
$dumpfile("memooriaROM_TB.vcd");
$dumpvars(0, memooriaROM_TB);
// # 100 $display("FIN de la simulacion");
// $finish;
end
always@(Clk)
#5 Clk <= ~Clk;
endmodule
```

Microprocesador_TB.v

```
module Microprocesador_TB(
    );

    reg Clk;
    reg Rst;
    reg [7:0] Datos_Entrada;
    reg [8:0] Instrucciones;
    wire [7:0] Direccion_Instrucciones;
    wire [7:0] Direccion_Datos;
    wire [7:0] Salida_Datos;
    wire RW;

    Microprocesador DUT (
        .Clk (Clk),
        .Rst (Rst),
        .Datos_Entrada (Datos_Entrada),
        .Instrucciones (Instrucciones),
        .Direccion_Instrucciones (Direccion_Instrucciones),
        .Direccion_Datos (Direccion_Datos),
        .Salida_Datos (Salida_Datos),
        .RW (RW)
    );

  initial
  begin
    Clk <= 0;
    Rst <= 1;
    Datos_Entrada <= 8'b00001111;
    Instrucciones <= 9'b000101011;
    #20 Rst <= 0;

    Datos_Entrada <= 8'b00001110;
    Instrucciones <= 9'b000101111;
    #10;
    Datos_Entrada <= 8'b00001110;
    Instrucciones <= 9'b001111011;
    #10;
    Datos_Entrada <= 8'b00001110;
    Instrucciones <= 9'b010001000;
    #10;
    Datos_Entrada <= 8'b00001110;
    Instrucciones <= 9'b001100101;
    #10;
    Datos_Entrada <= 8'b00001110;
    Instrucciones <= 9'b001011110;
    #10;
    Datos_Entrada <= 8'b00001110;
    Instrucciones <= 9'b110011000;
```

```
        #10;
        Datos_Entrada <= 8'b00001110;
        Instrucciones <= 9'b101010000;
        #10;
        Datos_Entrada <= 8'b00001110;
        Instrucciones <= 9'b110111001;
        #10;
        Datos_Entrada <= 8'b00110010;
        Instrucciones <= 9'b010101001;
        #10;
        Datos_Entrada <= 8'b00110010;
        Instrucciones <= 9'b011111010;
        #10;
        Datos_Entrada <= 8'b00110010;
        Instrucciones <= 9'b110001100;
        #10;
        Datos_Entrada <= 8'b00110010;
        Instrucciones <= 9'b111101100;
        #10;
        Datos_Entrada <= 8'b00110010;
        Instrucciones <= 9'b100011000;
        #10;
        Datos_Entrada <= 8'b00110010;
        Instrucciones <= 9'b000111111;
        #10;
        Datos_Entrada <= 8'b11111111;
        Instrucciones <= 9'b010001101;
        #10;
        Datos_Entrada <= 8'b11111111;
        Instrucciones <= 9'b111110001;
        #10;
        Datos_Entrada <= 8'b11111111;
        Instrucciones <= 9'b000111010;
        #10;
    end

    always@(Clk) begin
        #5 Clk <= ~Clk;
    end

endmodule
```

ALU_TB.v

```
module ALU_TB(

    );
  reg [3:0] Control;
  reg [15:0] Operandos;
  wire [7:0] Resultado;
  wire [2:0] Banderas;

  ALU DUT(
    .i_Control_ALU(Control),
    .i_Operandos(Operandos),
    .o_Resultado(Resultado),
    .o_Banderas_Estado(Banderas)
  );

  initial
  begin
    Control <= 0;
    Operandos <= 0;
    #20;
    Control <= 4'b1000;
    Operandos[15:8] <= 8'b01111111;
    Operandos[7:0] <= 8'b00000001;
    #20;
    Control <= 4'b1001;
    Operandos[15:8] <= 8'b11101110;
    Operandos[7:0] <= 8'b11110111;
    #20;
    Control <= 4'b1010;
    Operandos[15:8] <= 8'b00000101;
    Operandos[7:0] <= 8'b00000011;
    #20;
    Control <= 4'b1011;
    Operandos[15:8] <= 8'b00000101;
    Operandos[7:0] <= 8'b00000011;
    #20;
    Control <= 4'b1100;
    Operandos[15:8] <= 8'b01010101;
    Operandos[7:0] <= 8'b01010101;
    #20;
    Control <= 4'b1101;
    Operandos[15:8] <= 8'b01010101;
    Operandos[7:0] <= 8'b10101010;
    #20;
    Control <= 4'b1110;
    Operandos[15:8] <= 8'b01010101;
    Operandos[7:0] <= 8'b10101010;
    #20;
    Control <= 4'b1111;
    Operandos[15:8] <= 8'b00000101;
    Operandos[7:0] <= 8'b00000010;
```

```
    #20;
    Control <= 4'b0010;
    Operandos[15:8] <= 8'b00000101;
    Operandos[7:0] <= 8'b00000011;
    #20;
  end
endmodule
```

Control_TB.v

```
module Control_TB(
    );

    reg [8:0] Instrucciones;
    wire PC;
    wire Registros;
    wire [3:0] ALU;
    wire [4:0] Direccionamiento;
    wire [3:0] Saltos;

    Control DUT (
        .i_Instrucciones(Instrucciones),
        .o_Control_PC(PC),
        .o_Control_Registros(Registros),
        .o_Control_ALU(ALU),
        .o_Control_Direccionamiento(Direccionamiento),
        .o_Control_Saltos(Saltos)
    );

    initial begin

        Instrucciones<=9'b010100101;
        #20;

        Instrucciones<=9'b010100101;
        #20;
        Instrucciones<=9'b100010101;
        #20;
        Instrucciones<=9'b111000110;
        #20;
        Instrucciones<=9'b110010100;
        #20;
        Instrucciones<=9'b000111111;
        #20;
    end


endmodule
```

PC_TB.v

```
module PC_TB(
    );
    reg Clk;
    reg Rst;
    reg Control;
    reg [1:0] Saltos;
    reg [7:0] Direccion_Salto;
    wire [7:0] Direccion_Instrucciones;
    wire [7:0] Direccion_PC;

    PC DUT(
        .i_Clk(Clk),
        .i_Rst(Rst),
        .i_Control_PC(Control),
        .i_Control_Saltos(Saltos),
        .i_Direccion_Salto(Direccion_Salto),
        .o_Direccion_Instrucciones(Direccion_Instrucciones),
        .o_Direccion_PC(Direccion_PC)
    );

initial
begin
    Clk<=0;
    Rst<=1;
    Control<=1'b1; Saltos<=2'b00; Direccion_Salto<=8'b00000001;
    #20 Rst<=0;
    Control<=1'b1; Saltos<=2'b00; Direccion_Salto<=8'b00000001;
    #20;
    Control<=1'b1; Saltos<=2'b10; Direccion_Salto<=8'b11111111;
    #20;
    Control<=1'b1; Saltos<=2'b01; Direccion_Salto<=8'b01111000;
    #20;
    Control<=1'b0; Saltos<=2'b00; Direccion_Salto<=8'b01010101;
    #20;
    Control<=1'b0; Saltos<=2'b01; Direccion_Salto<=8'b11111111;
    #20;
    Control<=1'b0; Saltos<=2'b10; Direccion_Salto<=8'b10101011;
    #20;
    Control<=1'b1; Saltos<=2'b00; Direccion_Salto<=8'b01010101;
    #20;
end

always@(Clk)
  #5 Clk <= ~Clk;

endmodule
```

Registros_TB.v

```
module Registros_TB (
    );
    reg Rst;
    reg Clk;
    reg [7:0] PC;
    reg [7:0] ALU;
    reg [7:0] Datos_Entrada;
    reg Control;
    reg [8:0] Instrucciones;
    wire [15:0] Direccionamiento;
    wire [7:0] oPC;
    wire [15:0] oALU;

    Registros DUT(
        .i_Rst (Rst),
        .i_Clk (Clk),
        .i_Direccion_PC (PC),
        .i_Resultado_ALU (ALU),
        .i_Control_Registros (Control),
        .i_Datos_Entrada (Datos_Entrada),
        .i_Instrucciones (Instrucciones),
        .o_DireccionDato (Direccionamiento),
        .o_Direccion_Salto (oPC),
        .o_Operandos (oALU)
    );

    initial
    begin
        Rst <= 1;
        Clk <= 0;
        PC <= 8'b01010101;
        ALU <= 8'b10101010;
        Control <= 1;
        Datos_Entrada <= 8'b11111111;
        Instrucciones <= 9'b010101011;
        #20 Rst <= 0;

        Control <= 3'bx;
        Instrucciones <= 9'b001100111;
        #20;
        Control <= 0;
        Instrucciones <= 9'b010001000;
        #20;
        Control <= 1;
        Instrucciones <= 9'b101011100;
        #20;
        Control <= 0;
        Instrucciones <= 9'b100100101;
```

```
        #20;
        Control <= 1;
        Instrucciones <= 9'b110111110;
        #20;
        Control <= 1'bx;
        Instrucciones <= 9'b111101000;
        #20;
    end

    always@(Clk) begin
        #5 Clk <= ~Clk;
    end
endmodule
```

Saltos_TB.v

```
module Saltos_TB(
    );
    reg [3:0] Control;
    reg [2:0] Estado;
    wire [1:0] PC;

    Saltos DUT (
        .i_Control_Salto(Control),
        .i_BanEstado(Estado),
        .o_Salto_PC(PC)
    );

    initial begin
        Control<=4'b0000;
        Estado<=3'b000;
        #20;
        Control<=4'b0001;
        Estado<=3'b001;
        #20;
        Control<=4'b0010;
        Estado<=3'b010;
        #20;
        Control<=4'b0011;
        Estado<=3'b011;
        #20;
        Control<=4'b0100;
        Estado<=3'b100;
        #20;
        Control<=4'b0101;
        Estado<=3'b101;
        #20;
        Control<=4'b0110;
        Estado<=3'b110;
        #20;
        Control<=4'b0111;
        Estado<=3'b111;
        #20;
        Control<=4'b1000;
        Estado<=3'b000;
        #20;
        Control<=4'b1001;
        Estado<=3'b001;
        #20;
        Control<=4'b1010;
        Estado<=3'b010;
        #20;
        Control<=4'b1011;
        Estado<=3'b001;
```

```
            #20;
            Control<=4'b1100;
            Estado<=3'b100;
            #20;
            Control<=4'b1101;
            Estado<=3'b010;
            #20;
            Control<=4'b1110;
            Estado<=3'b001;
            #20;
            Control<=4'b1111;
            Estado<=3'b010;
            #20;
        end


endmodule
```

Direccionamiento_TB.v

```
module Direccionamiento_TB(
    );
 reg [4:0] i_Control_Direc;
 reg [15:0] i_DireccionDato;
 wire [7:0] o_Direccion_Datos;
 wire [7:0] o_Salida_Datos;
 wire o_RW;

 Direccionamiento DUT(
     .i_Control_Direc (i_Control_Direc),
     .i_DireccionDato (i_DireccionDato),
     .o_Direccion_Datos (o_Direccion_Datos),
     .o_Salida_Datos (o_Salida_Datos),
     .o_RW (o_RW)
 );

  initial
  begin
    i_Control_Direc <= 0;
    i_DireccionDato <= 0;
    #20;

    i_Control_Direc <= 5'b01101;
    i_DireccionDato[15:8] <= 8'b10101010;
    i_DireccionDato[7:0] <= 8'b01010101;
    #20;
    i_Control_Direc <= 5'b10111;
    i_DireccionDato[15:8] <= 8'b00111010;
    i_DireccionDato[7:0] <= 8'b01011100;
    #20;
    i_Control_Direc <= 5'b11011;
    i_DireccionDato[15:8] <= 8'b11110000;
    i_DireccionDato[7:0] <= 8'b01111101;
    #20;
    i_Control_Direc <= 5'b00101;
    i_DireccionDato[15:8] <= 8'b00000011;
    i_DireccionDato[7:0] <= 8'b11111000;
    #20;
    i_Control_Direc <= 5'b01000;
    i_DireccionDato[15:8] <= 8'b00011100;
    i_DireccionDato[7:0] <= 8'b00001111;
    #20;
    i_Control_Direc <= 5'b10001;
    i_DireccionDato[15:8] <= 8'b01100111;
    i_DireccionDato[7:0] <= 8'b011010100;
    #20;
  end
endmodule
```