

Universidad Autónoma de Zacatecas

Unidad Académica de Eléctrica

Ingeniería en Robótica y Mecatrónica



Microprocesador

Doctor Remberto Sandoval Arechiga

	Grupo	Fecha
Juan Bernardo Esparza Luevano		24/11/2020
Janelly de Jesús Aguilar Zamora	5 "A"	
Ricardo Heredia Dávila		
Leonardo Daniel González Bribiesca		

RESUMEN

Un microprocesador es un circuito electrónico que procesa la energía necesaria para que un dispositivo electrónico en el que se encuentra funcione, ejecutando los comandos y los programas adecuadamente; Este componente electrónico es el encargado de procesar y ejecutar las instrucciones codificadas en números binarios.

El microprocesador se compone básicamente de registros, una unidad de control, una unidad aritmético lógica (ALU) principalmente.

Hoy en día un microprocesador es capaz de recibir las instrucciones, decodificarlas, buscar los programas compatibles para ejecutarlas, las ejecuta, analiza los datos y muestra los resultados de dicho proceso en 1 segundo o menos.

Todos nuestros diagramas pueden ser consultados en:

<https://drive.google.com/file/d/1J68JCfBkgZsLo5WMbTPKHmnF9pdZQz0O/view?usp=sharing>

ÍNDICE

RESUMEN	2
ÍNDICE	3
INTRODUCCIÓN	4
ARQUITECTURA	5
ESQUEMÁTICOS	28
SIMULACIÓN	29
CONCLUSIONES	30
APÉNDICE A (Código)	31
APÉNDICE B (Testbench)	38

Introducción

En el presente documento plasmamos a detalle todo el proceso que se llevó a cabo así como los resultados y funcionamiento del microprocesador RISC que realizamos con arquitectura harvard (Arquitectura con memoria de programa y de datos separadas y solo accesibles a través de buses distintos) y el apoyo de la herramienta Vivado (Utilizando el lenguaje Verilog HDL) en el curso de Microcontroladores, el cual será capaz de sumar y restar, incluimos también una memoria RAM y una memoria ROM para así poder efectuar multiplicaciones creando una computadora (Máquina electrónica capaz de almacenar información y tratarla automáticamente mediante operaciones matemáticas y lógicas controladas por programas informáticos.).

La arquitectura Harvard es una arquitectura de computadora con pistas de almacenamiento y de señal físicamente separadas para las instrucciones y para los datos. Esta arquitectura ofrece la posibilidad de poder acceder a una sola instrucción en un ciclo de reloj. Mientras la memoria de programa es accedida la memoria de datos está en un bus independiente y puede ser leída y escrita. Esta separación de buses permite que una instrucción sea ejecutada mientras la siguiente es extraída.

Este microprocesador contará con un set de 8 instrucciones, de esas ocho a dos les llamaremos “LOAD” las cuales se encargarán, a grandes rasgos, de cargar datos en registros, a otras dos instrucciones las llamaremos “STORE” las cuales se encargan de almacenar información en direcciones de memoria , otra instrucción será “MOVE” la cual consiste en mover información de un registro RY a un registro RX, otra instrucción será la de “MATH” esta instrucción es en la que se realizan las operaciones matemáticas con RX y el resultado se almacena en R0 las últimas dos instrucciones son la JUMP y la NOP de las cuales “JUMP” salta del pc a una dirección de memoria en ciertas condiciones y “NOP” es la instrucción que se ejecuta cuando no se está haciendo ninguna de las anteriores mencionadas y se puede describir como no operación.

ARQUITECTURA

Caja negra

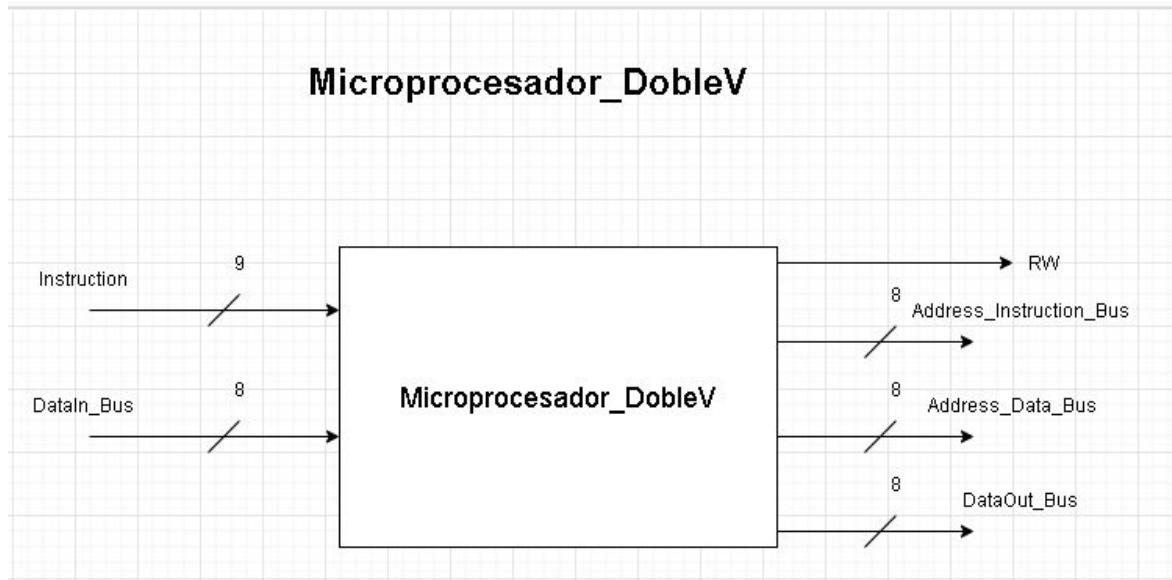


Figura 1.1. Diagrama nivel Top del Microprocesador_DobleV.

Entradas:

Nombre de la señal	Tamaño	Descripción
Instruction	9	Entrada de 9 bits que indica la instrucción a realizar
DataIn_Bus	8	Entrada de 8 bits que transmite la información a manipular al microprocesador

Figura 1.2. Tabla con el nombre y descripción de las señales de entrada.

Salidas:

Nombre de la señal	Tamaño	Descripción
Address_Instruction_Bus	8	Salida de 8 bits que guarda la instrucción de operación en un bus de datos.
* Address_Data_Bus	8	Salida de 8 bits que transfiere datos entre una ubicación de memoria, ubicada dentro del segmento de datos, y AL
DataOut_Bus	8	Salida de 8 bits que transmite la información a una memoria
RW	1	Salida de 1 bit que nos dice si el microprocesador está leyendo o escribiendo información

Figura 1.3.Tabla con el nombre y descripción de las señales de salida.

Set de Instrucciones

Instruction	Arguments	Description	Comments
LOAD	RX,#NUM	Load #Num to register X	#Num is 3 bits [0,7]
LOAD	RX,[RY]	Load data at address [RY] from memory	RY and RX are 3 bits[0,7]
STORE	#NUM	Store #Num to [RX] address memory	#Num is 3 bits [0,7]
STORE	[RX],RY	Stores data at Register RY in [RX] memory address	RY and RX are 3 bits [0,7]
MOVE	RX,RY	Move data form register RY to RX	RY and RX are 3 bits [0,7]
MATH	RX,OP	DO MATH OPERATION WITH RX, AND STORES RESULT IN R0	OP: 0: R0=R0+RX 1: R0=R0-RX 2: R0= R0<<RX 3: R0= R0>>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	

Caja Blanca

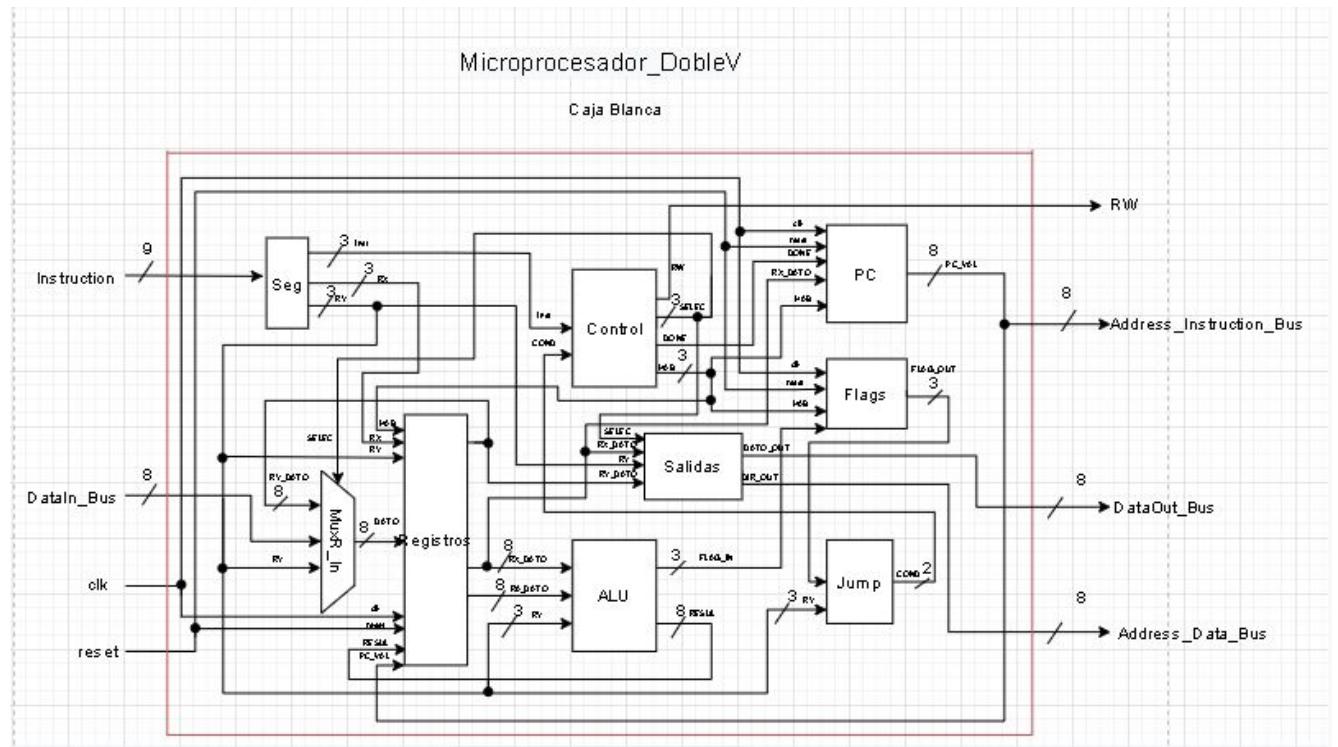


Figura 2.1. Diagrama del bloque completo de la caja negra, se puede observar cada uno de los bloques internos así como las señales internas.

Bloque	Descripción
Registros	Estos almacenan en datos y direcciones temporalmente. Estos registros sirven para suministrar información necesaria al realizar una instrucción.
Control	Sincroniza y procesa las instrucciones a realizar. Manda señales a diferentes bloques para llevar a cabo la instrucción que esté presente.
ALU	Unidad Aritmética Lógica, se encarga de realizar las operaciones matemáticas y lógicas que se requieran. Además de proporcionar el resultado también establece banderas en el caso de tener resultados como ceros, acarreos o signo negativo.
PC	Incrementa con cada instrucción que se realiza y apunta a la siguiente instrucción que se va a realizar.

Flags	Almacena temporalmente las banderas proporcionadas por la ALU para su requerimiento en otras instrucciones.
Seg	Segmenta la señal Instruction en 3 partes, la señal Inst de 3 bits (8:6), la señal RX de 3 bits (5:3) Y RY de 3 bits (2:0)
Jump	Bloque encargado de verificar si se ha realizado la condición especificada mediante el estado de las banderas, en el caso de que así sea se manda una señal al control para notificar que la condición se ha cumplido. También genera una señal para el caso específico de la condición número 1.
Salidas	Selecciona cual dato va a salir por el bus de datos y también cual dirección de datos va a salir por el bus de dirección.

Figura 2.2. Tabla con el nombre de cada bloque interno en la caja negra con su respectiva descripción.

Registros

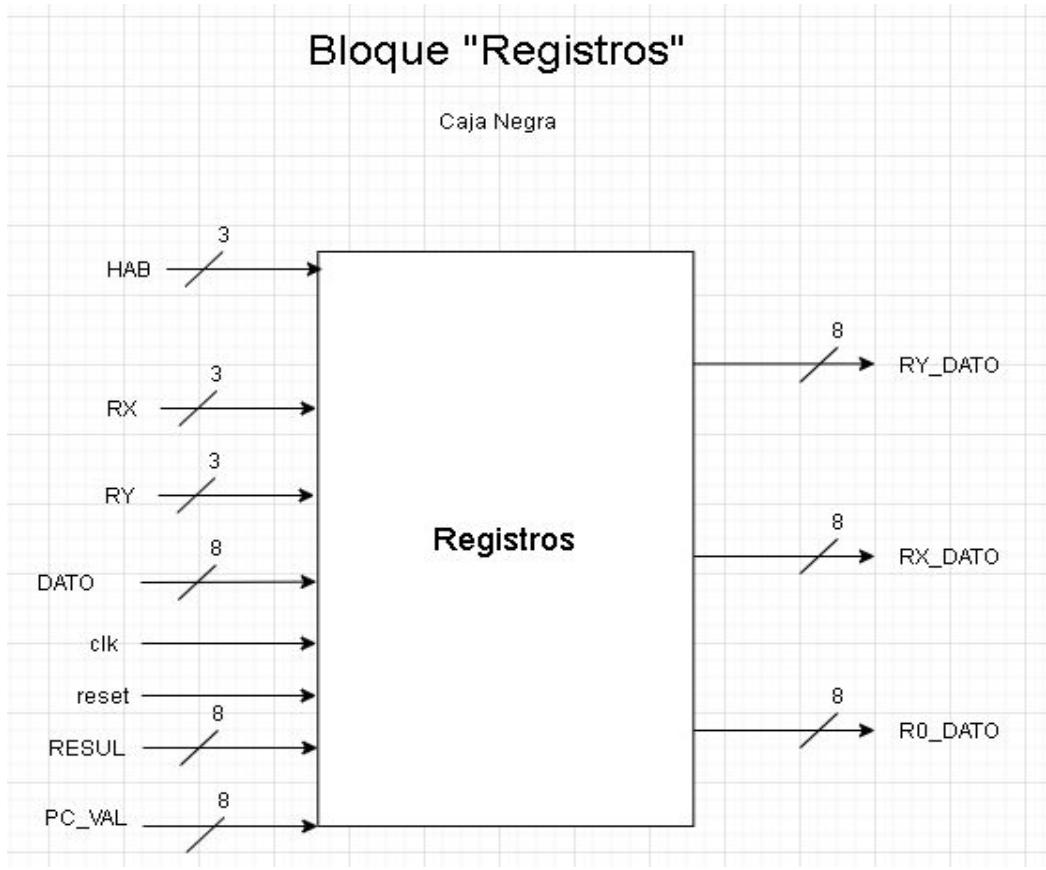
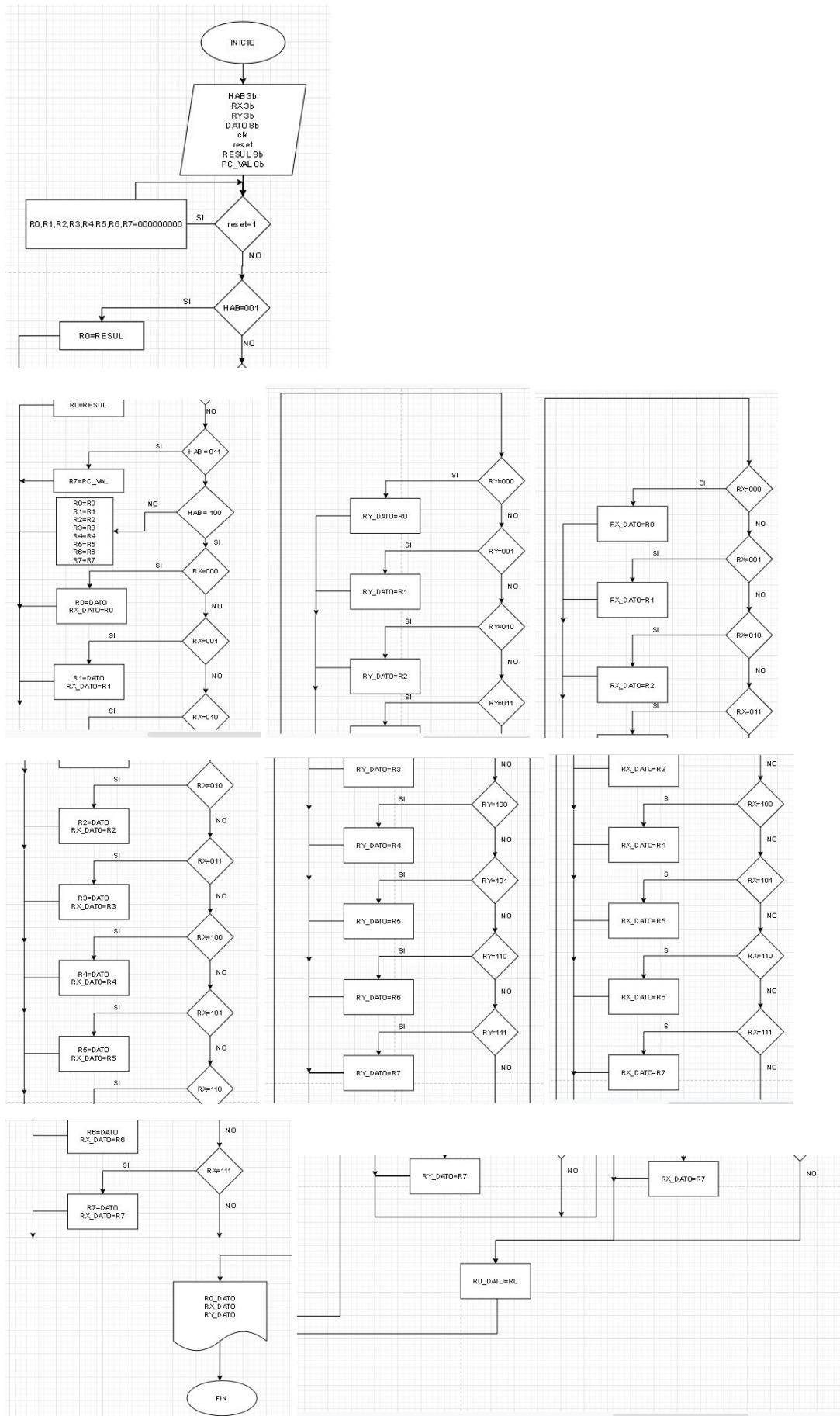


Figura 3.1. Diagrama del bloque de Registros.

Señal	Bits	Descripción	Señal	Bits	Descripción
HAB	2	Esta señal define si se va a guardar el resultado en el registro 0, y tambien si se va a guardar el pc en el registro 7.	rst	1	Esta señal reinicia el proceso del programa.
RX	3	Esta señal define cual de los 7 registros va a salir por la señal "RX_DATO", o bien guardar un dato en esa dirección	RESUL	8	Es el resultado de la alu previamente calculado.
RY	3	Esta señal define cual de los 7 registros va a salir por la señal "RY_DATO"	PC_VAL	8	Apunta a la instrucción que se va a realizar y se guarda como dato.
DATO	8	Esta señal puede tener varios datos dependiendo del multiplexor.	RY_DATO	8	Es la salida de 1 de los 7 registros , el registro que va a salir se define por RY
clk	1	Controla los ciclos de reloj.	RX_DATO	8	Es la salida de 1 de los 7 registros , el registro que va a salir se define por RX
rst	1	Esta señal reinicia el proceso del programa.	R0_DATO	8	Es la salida del registro R0.

Figura 3.2. Tabla con los nombres de las señales del bloque, así como sus número de bits y su respectiva descripción.

Descripción funcional



Control

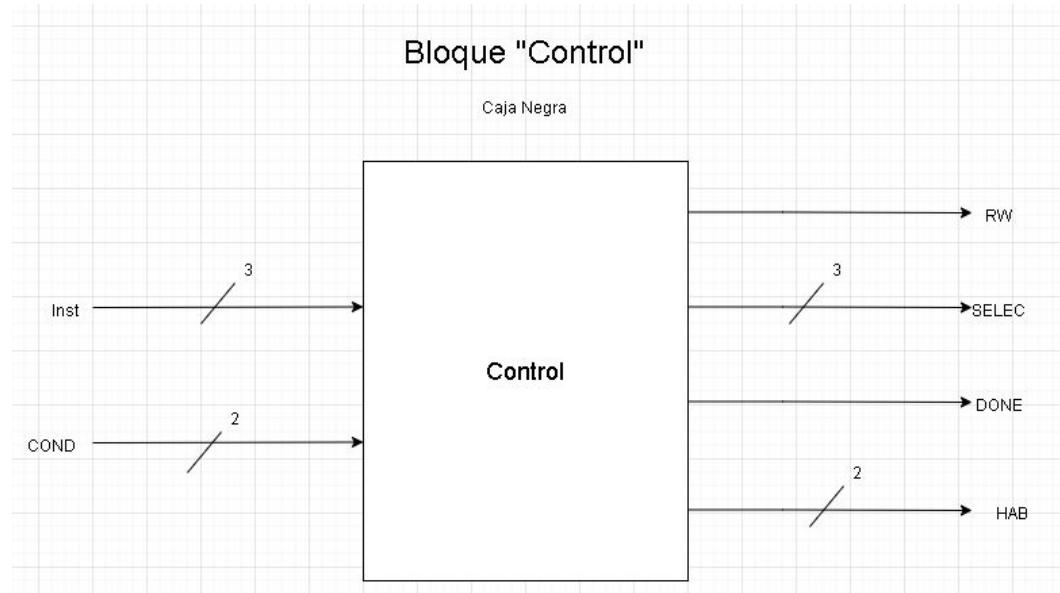
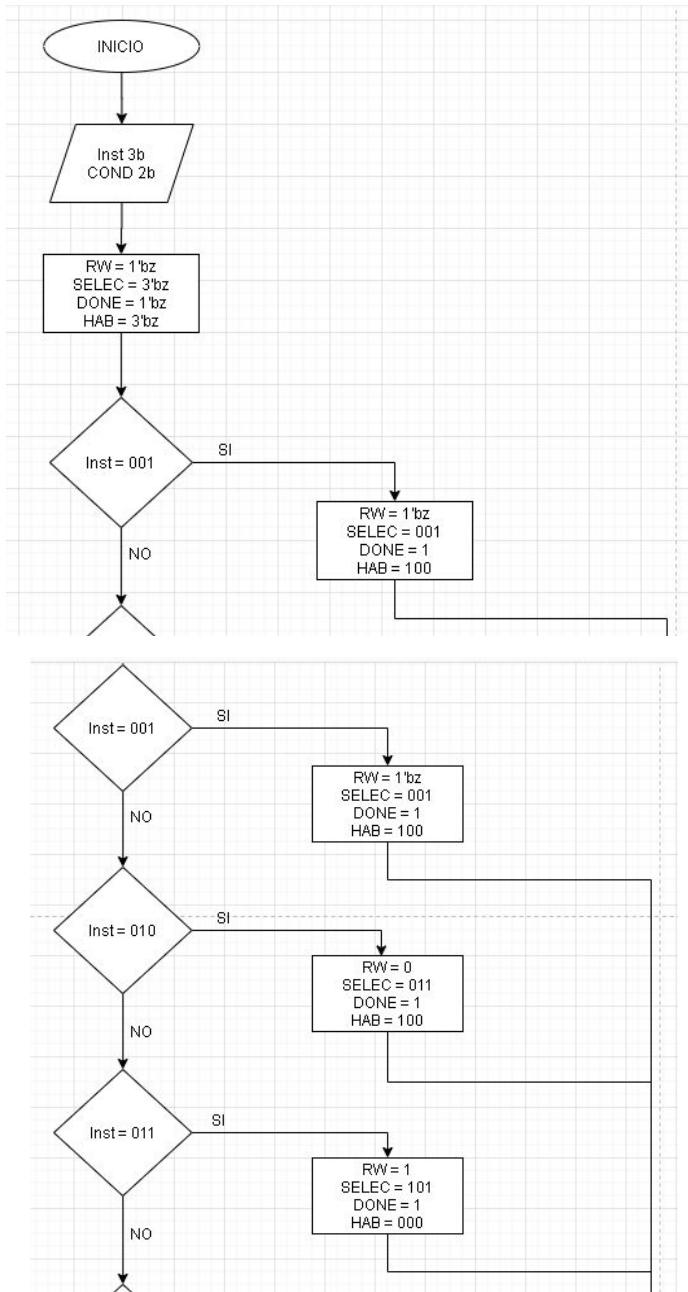


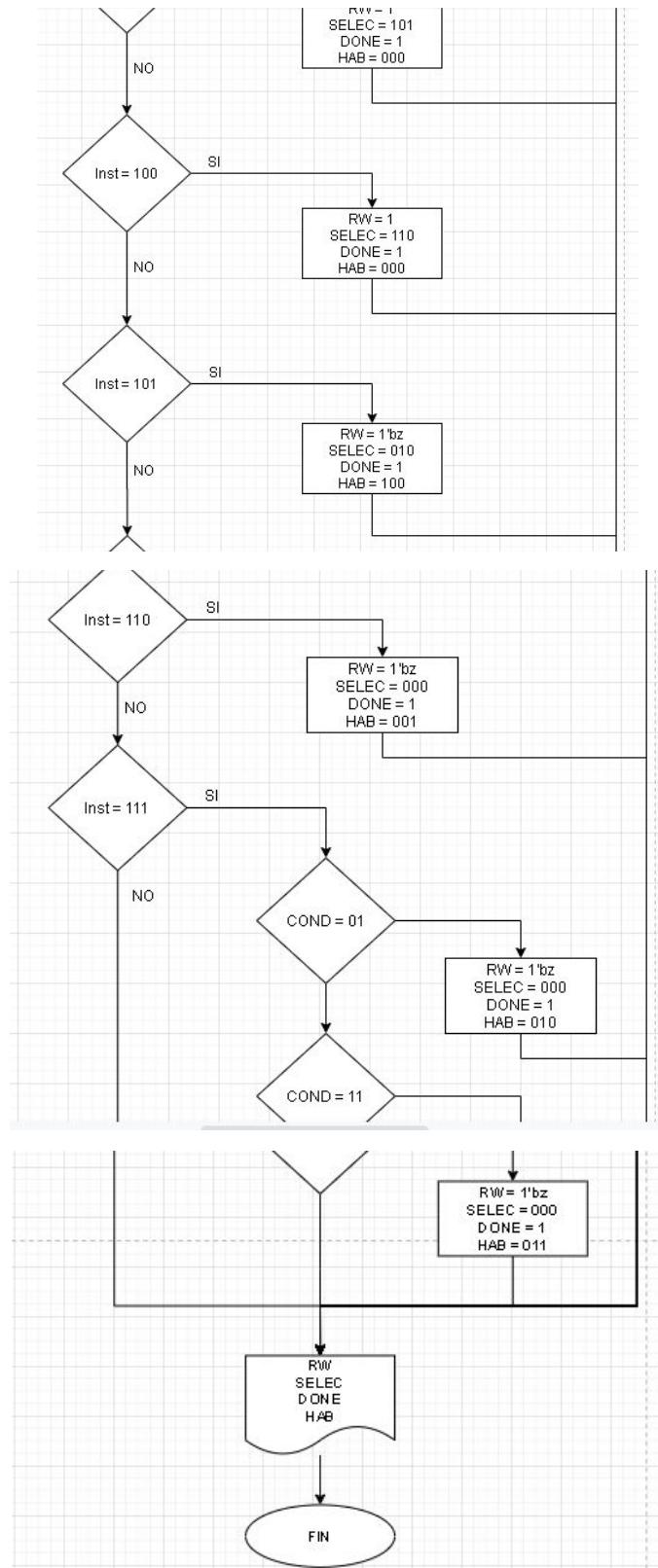
Figura 4.1. Diagrama del bloque de Control.

Señal	Bits	Descripción
Inst	3	Es la primera parte de la señal "Instruction" descompuesta, es de la longitud 8:6. Indica cual va a ser la instrucción del set de instrucciones que se va a realizar.
COND	2	Esta señal trae del jump, si la condición es verdadera o falsa
RW	1	Esta señal se utiliza para indicarle a la memoria si se va a leer o a escribir un dato. 0 para leer 1 para escribir
SELEC	3	Esta señal define cuál dato va a guardarse en registro y qué dato va a salir por DataOut_Bus y Address_Data_Bus.
DONE	1	Esta señal le indica al PC que ya se realizó una instrucción del set de instrucciones.
HAB	3	Según la instrucción, este habilita a los registros para guardar el resultado de los datos, la ALU y del PC, habilita las banderas y habilita si un dato saltará al PC

Figura 4.2. Tabla con los nombres de las señales del bloque, así como sus número de bits y su respectiva descripción.

Descripción funcional





Seg

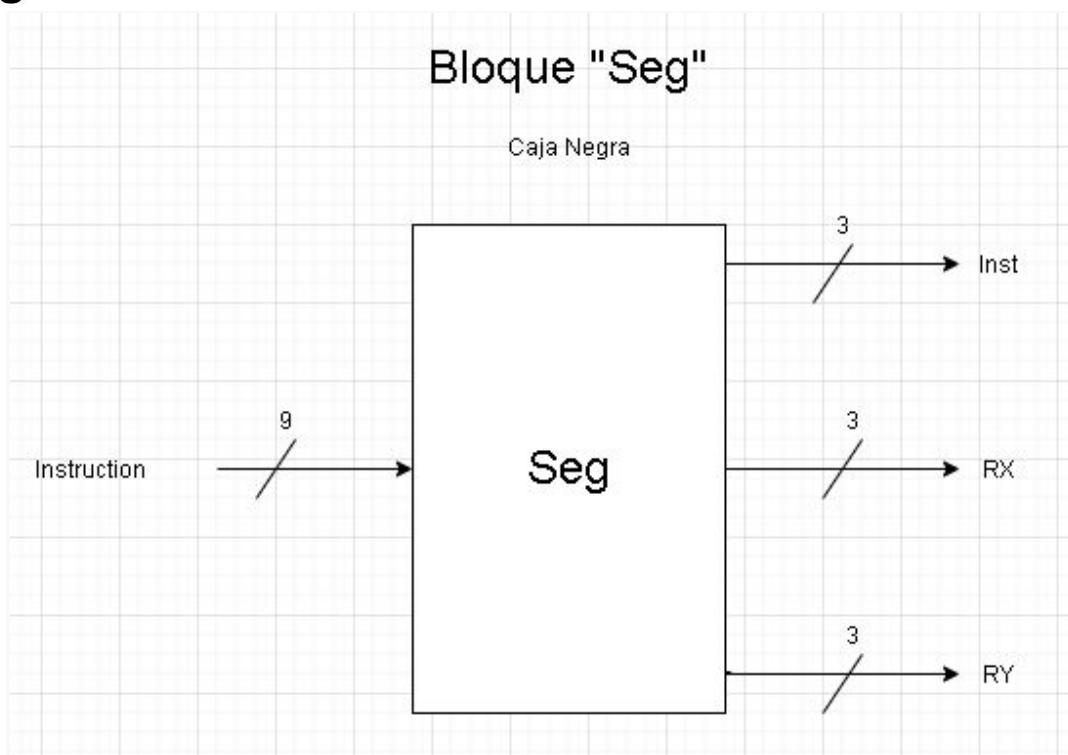


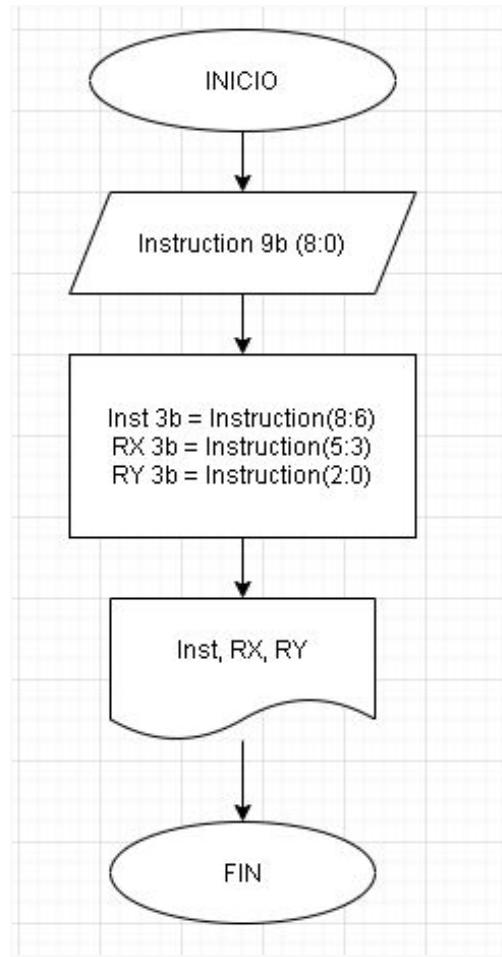
Figura 5.1. Diagrama del bloque de Seg.

Señal	Bits	Descripción
Instruction	9	Esta señal trae los datos que entran directo al microprocesador
Inst	3	Es la primera parte de la señal "Instruction" descompuesta, es de la longitud 8:6. Indica cual va a ser la instrucción del set de instrucciones que se va a realizar.
RX	3	Es la segunda parte de la señal "Instruction" descompuesta, su longitud es de 5:3 indica la dirección de registro
RY	3	Es la tercera parte de la señal "Instruction" descompuesta, su longitud es de 2:0. indica la dirección del registro, ademas de ser la señal de #NUM, Cont, OP.

Figura 5.2. Tabla con los nombres de las señales del bloque, así como sus número

de bits y su respectiva descripción.

Descripción funcional



PC

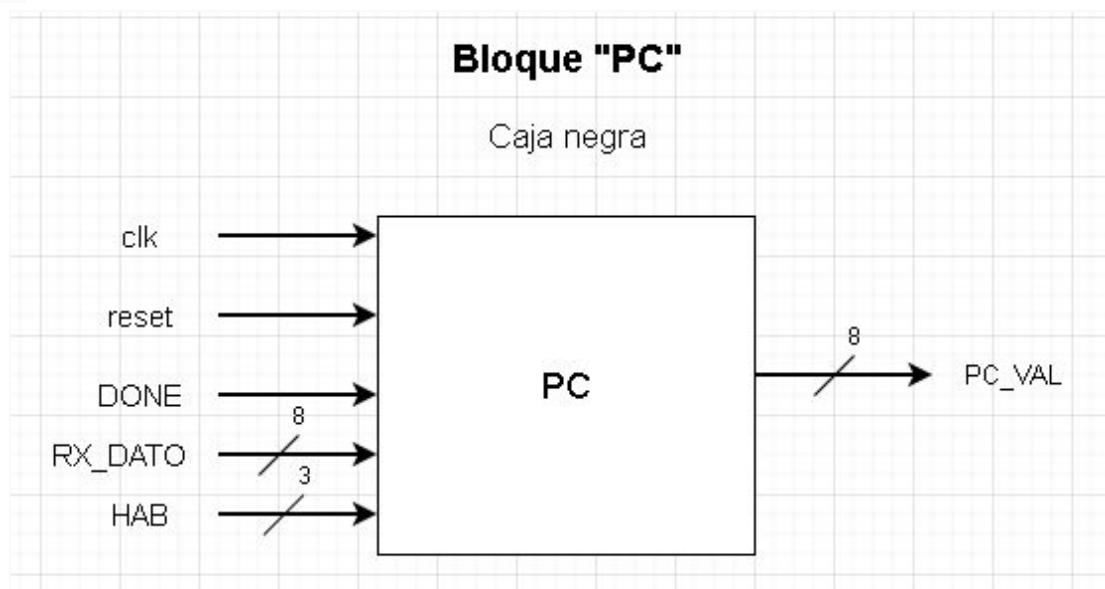
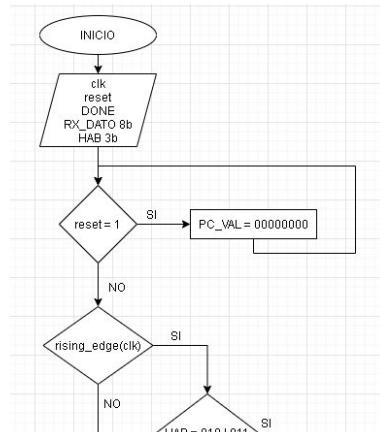


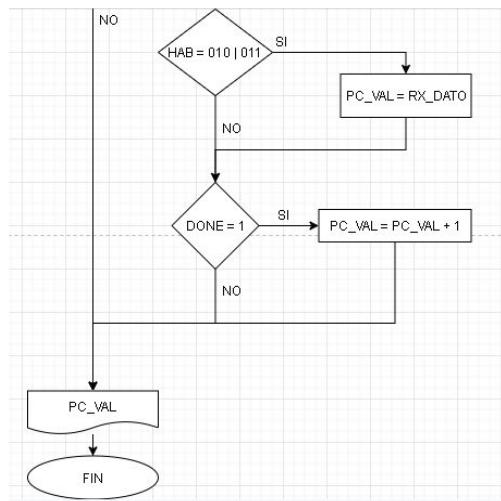
Figura 6.1. Diagrama del bloque de PC.

Señal	Bits	Descripción
DONE	1	Esta señal sirve para el control del conteo del numero de instrucciones ejecutadas
RX_DATO	8	Señal que almacena
HAB	3	Señal con la que se controla el dato que se cargara en la salida del pc
clk	1	Señal que controla los ciclos de reloj
reset	1	Señal que pone en un estado inicial al microcontrolador
PC_VAL	8	Señal que apunta a la siguiente instrucción dependiendo del contador interno

Figura 6.2. Tabla con los nombres de las señales del bloque, así como sus número de bits y su respectiva descripción.

Descripción funcional





Salidas

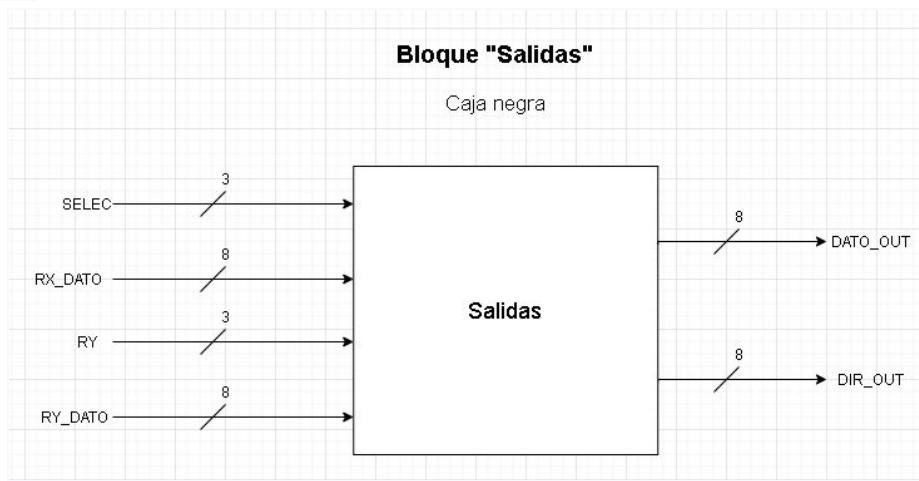
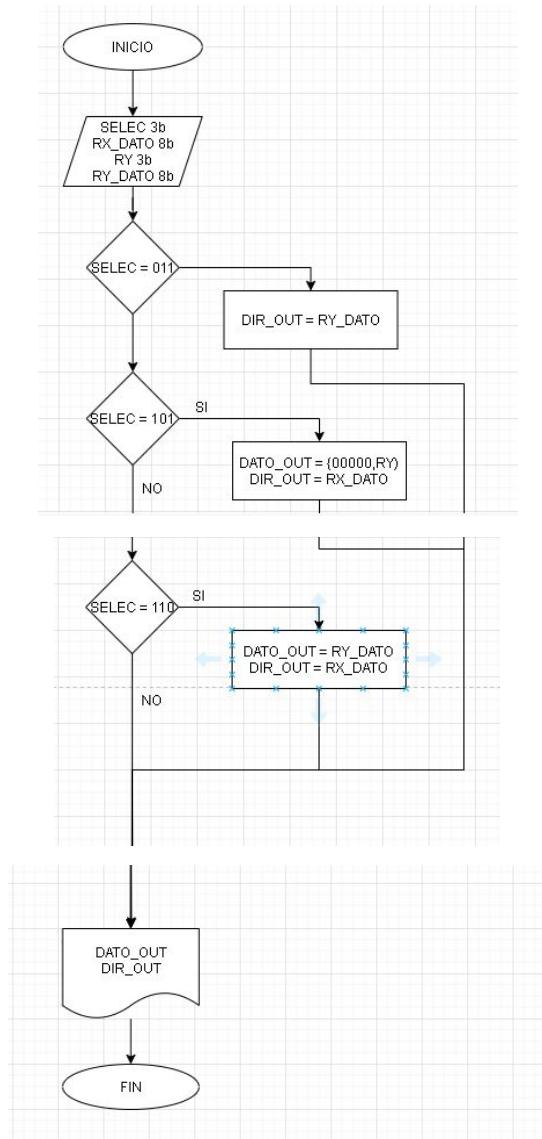


Figura 7.1. Diagrama del bloque de Salidas.

Señal	Bits	Descripción
SELEC	3	Controla los datos que saldrán del bloque
RX_DATO	8	Señal que almacena la dirección de memoria
RY	3	Señal que almacena datos necesarios para realizar las operaciones
RY_DATO	8	Señal que toma el valor de RY en ciertas condiciones
DATO_OUT	8	toma los valores de los tres bits menos significativos de RY y los otros 4 son ceros
DIR_OUT	8	Indica la dirección de memoria en la que se encuentra o era almacenado un dato

Figura 7.2. Tabla con los nombres de las señales del bloque, así como sus número de bits y su respectiva descripción.

Descripción funcional



ALU

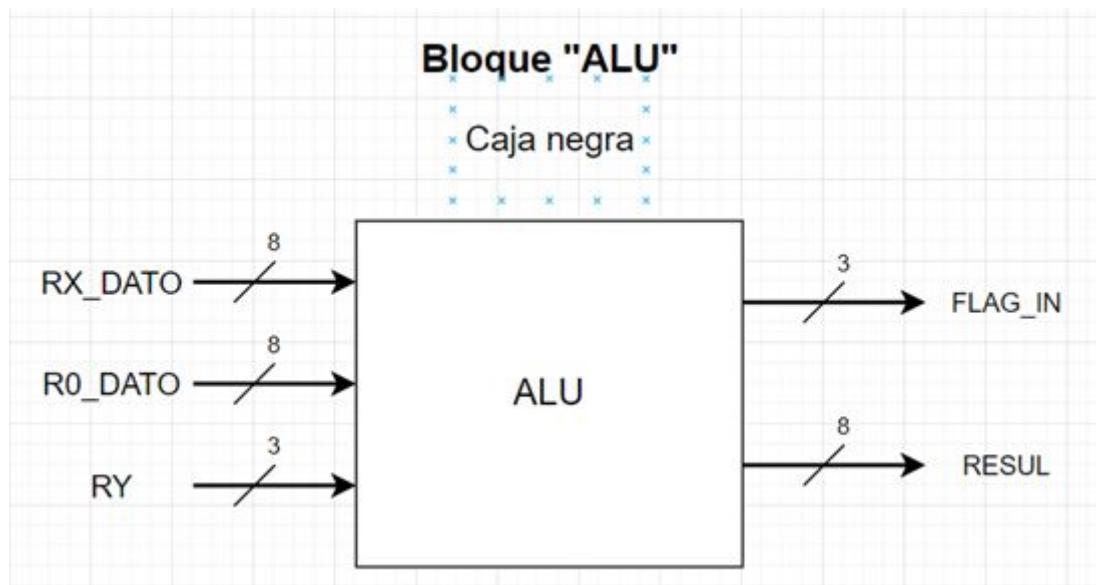
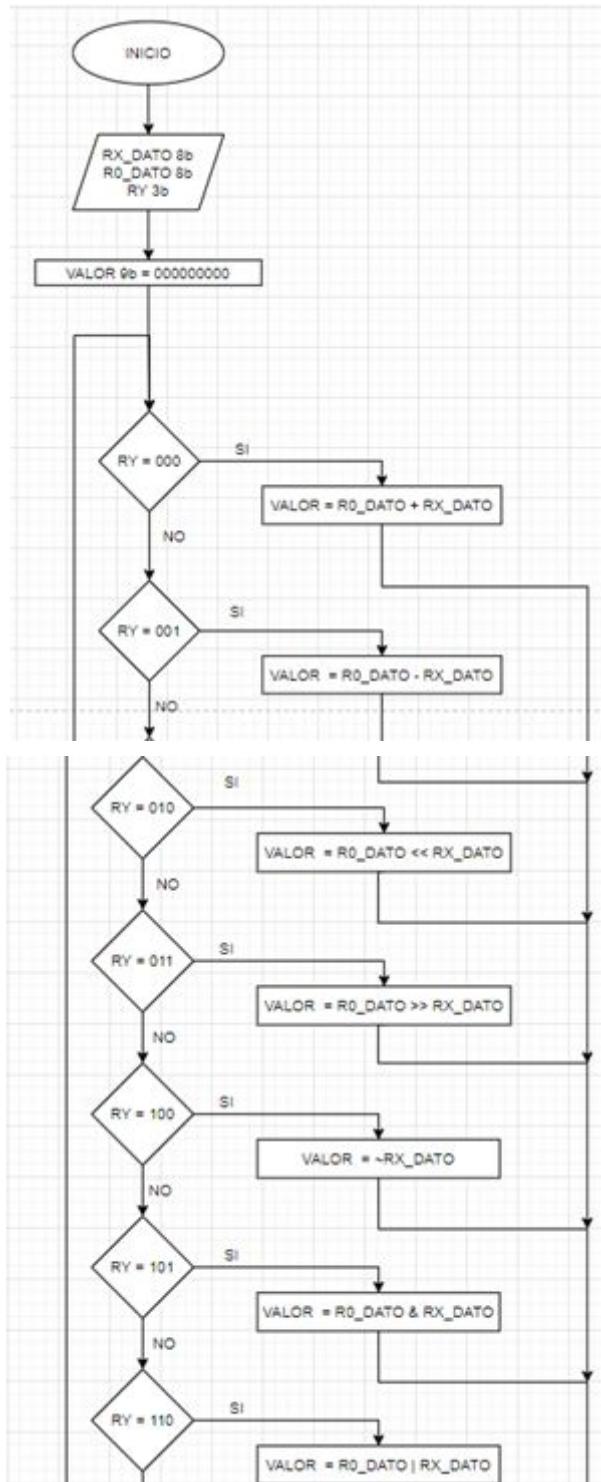
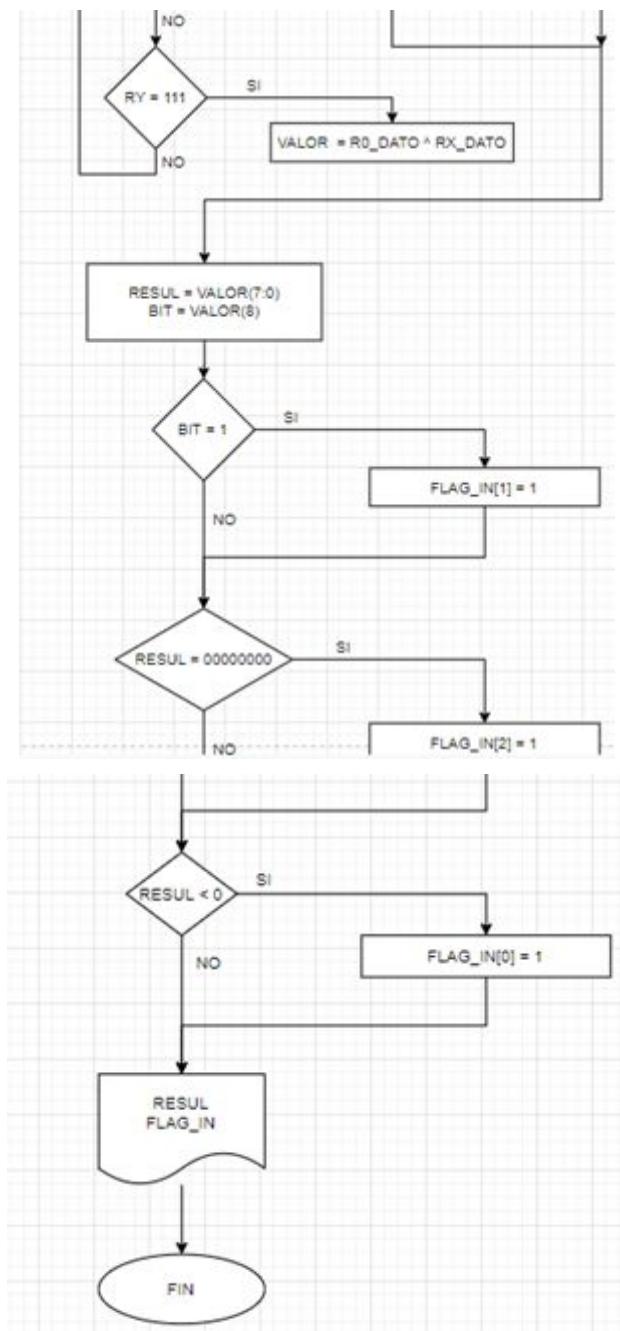


Figura 8.1. Diagrama del bloque de ALU

Descripción funcional





Señal	Tamaño	Descripción
RX_DATO	8	Entrada que cuenta con los datos del registro RX para operar
R0_DATO	8	Entrada con los datos del registro 0 para operar
* RY	3	Entrada que dicta qué operación realizar *
FLAG_IN	3	Salida que contiene las banderas generadas por las operaciones
RESUL	8	Salida con el resultado de la operación

Figura 8.2. Tabla con los nombres de las señales del bloque, así como sus número de bits y su respectiva descripción.

Jump

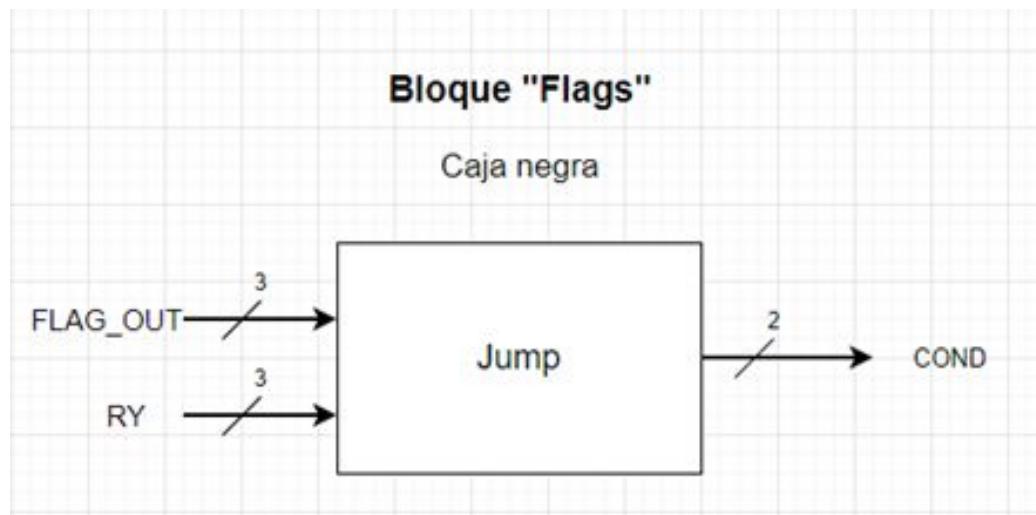
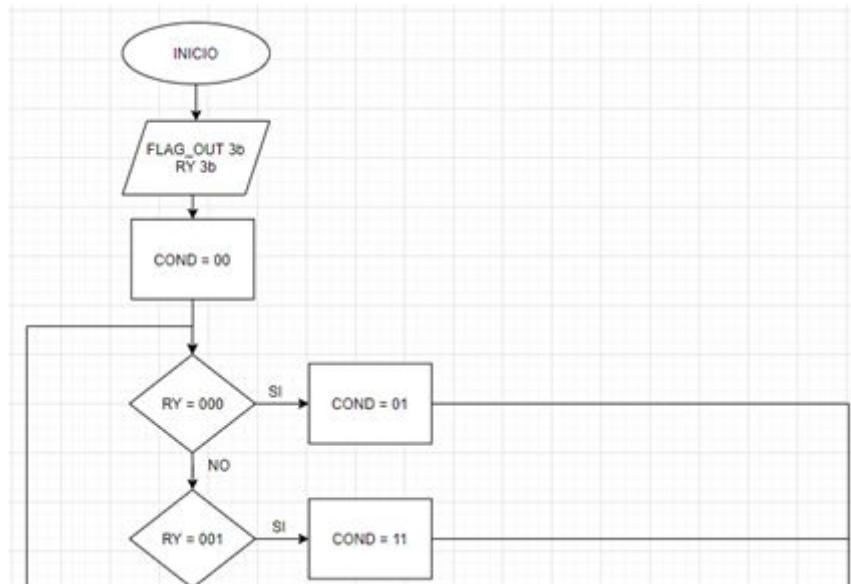
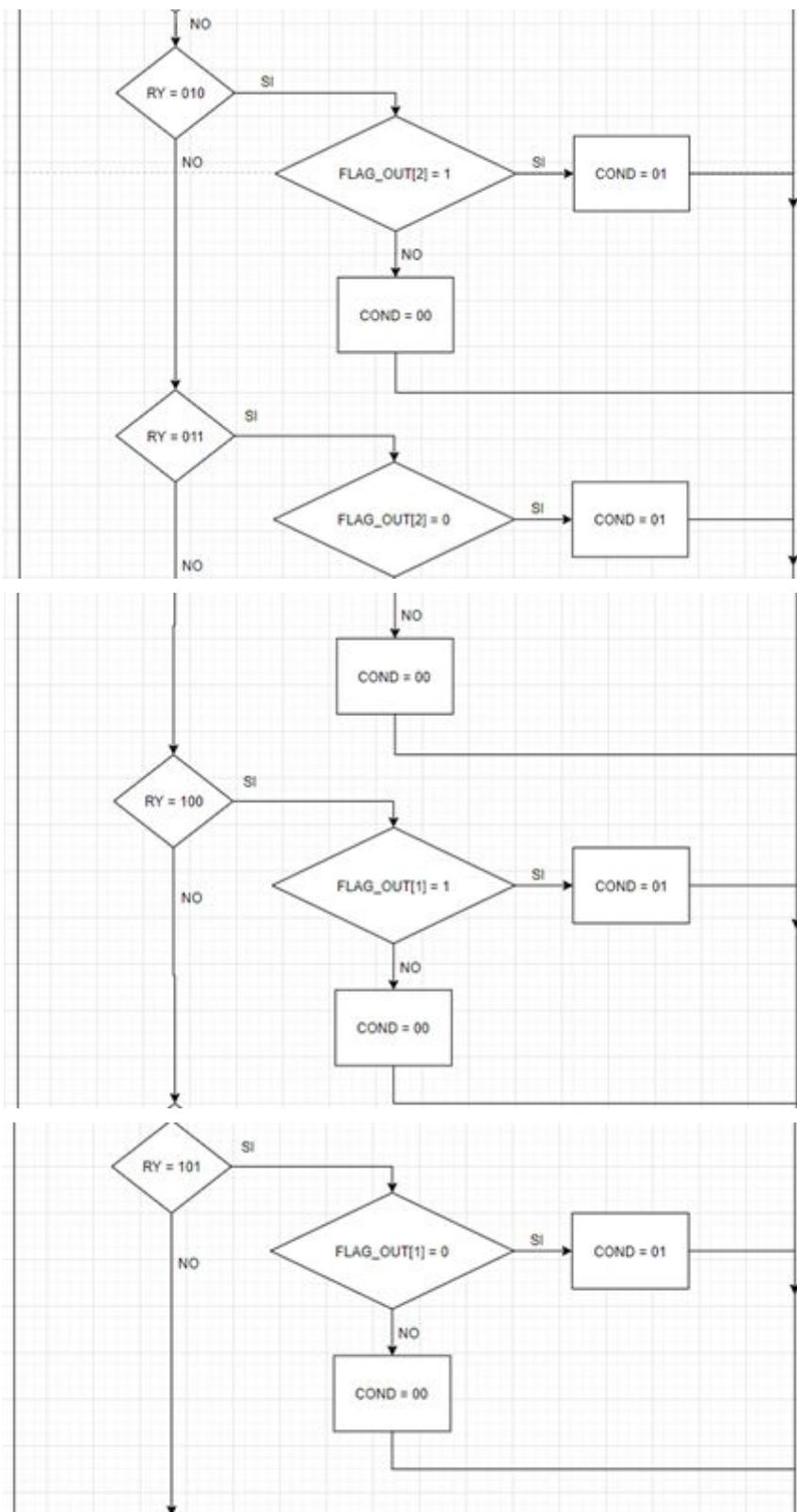
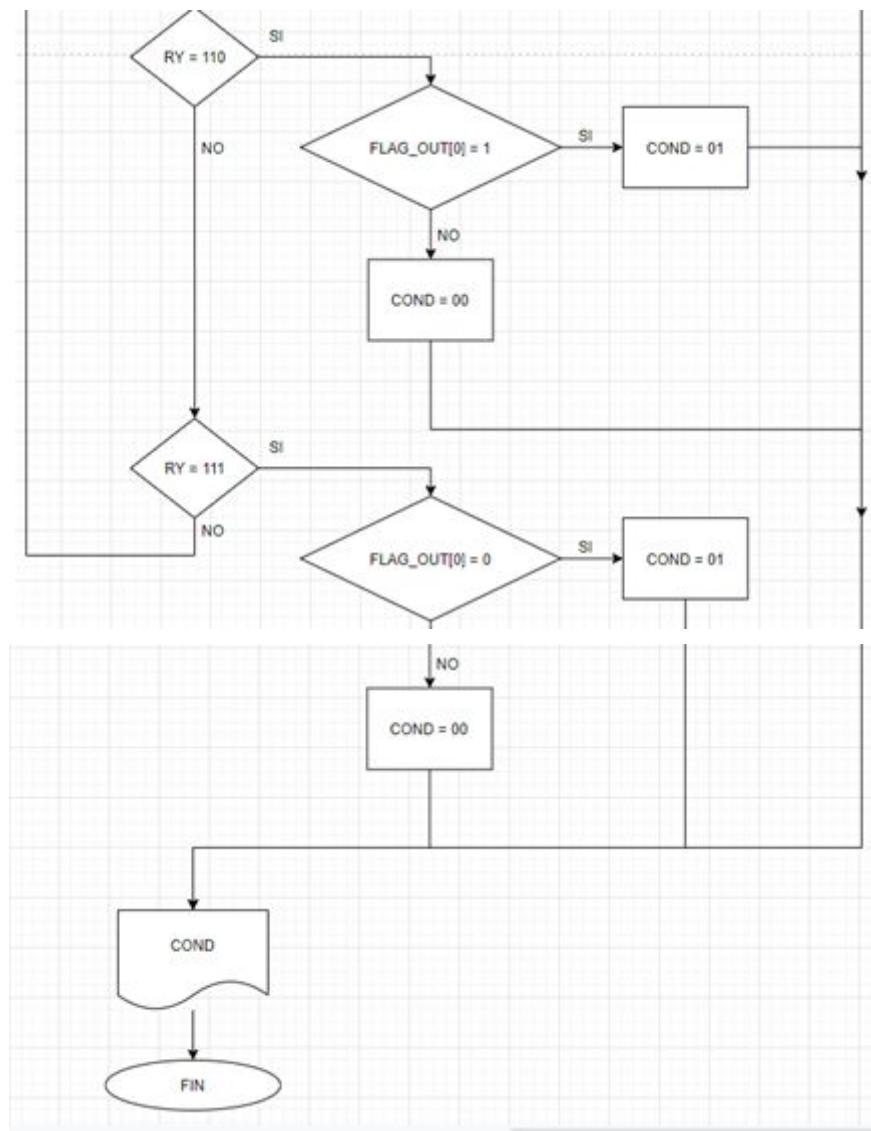


Figura 9.1. Diagrama del bloque Jump

Descripción funcional







Señal	Tamaño	Descripción
* FLAG_OUT	3	Entrada que cuenta con los datos de las banderas realizadas por el ALU (Guardadas desde la última operación MATH)
RY	3	Entrada con el dato de la condición a verificar
COND	2	Salida que habilita el Salto (JUMP)

Figura 9.2. Tabla con los nombres de las señales del bloque, así como sus número de bits y su respectiva descripción.

Flags

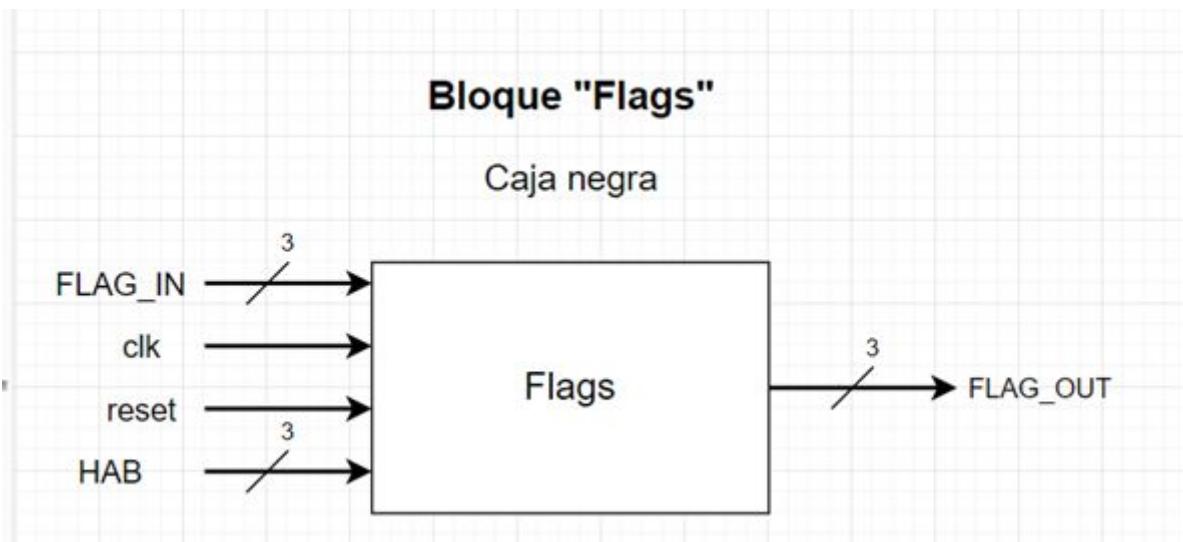
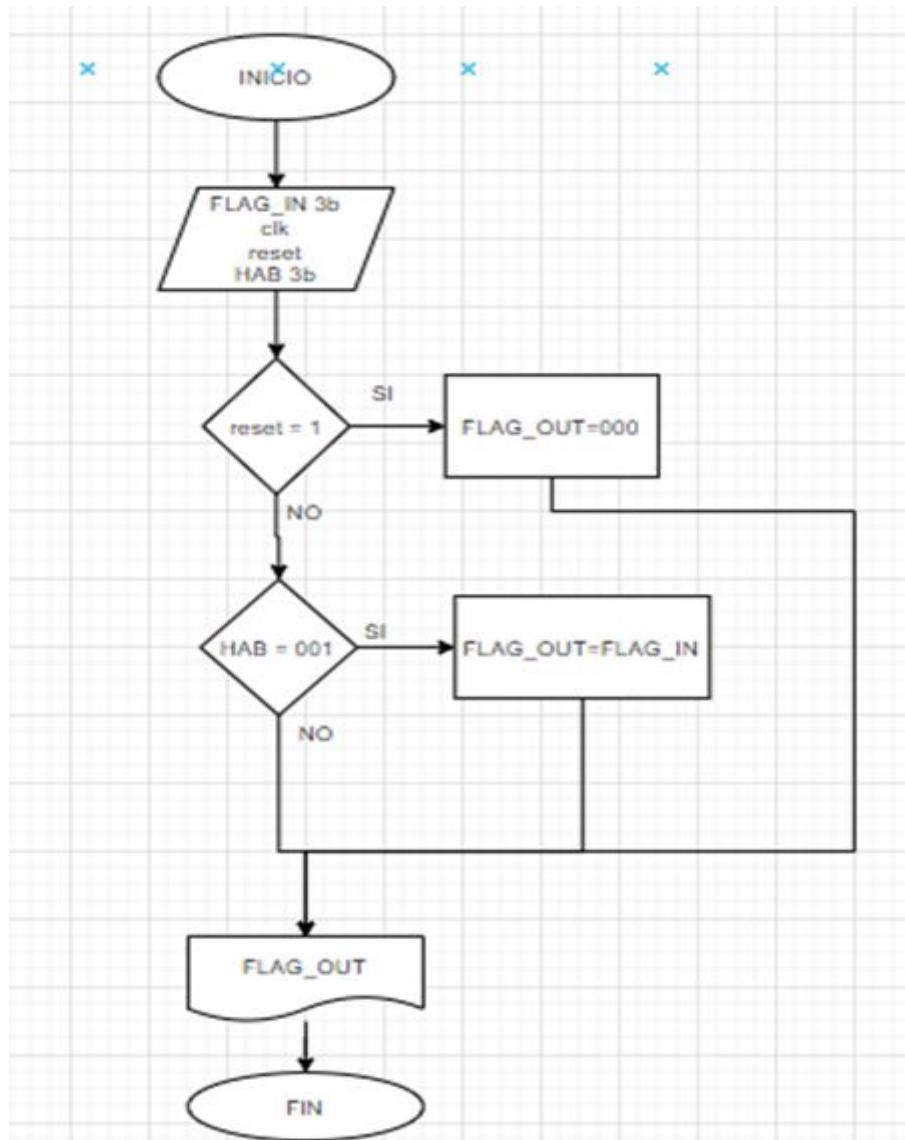


Figura 10.1. Diagrama del bloque Flags.

Descripción Funcional



Señal	Tamaño	Descripción
FLAG_IN	3	Entrada que cuenta con los datos de las banderas realizadas por el ALU (Instantáneamente)
clk	1	Entrada global del reloj
reset	1	Entrada para volver los datos almacenados a un estado inicial
HAB	3	Entrada que habilita guardar los datos en el registro
FLAG_OUT	3	Salida que cuenta con los datos de las banderas realizadas por el ALU (Guardadas desde la última operación MATH)

Figura 10.2. Tabla con los nombres de las señales del bloque, así como sus número de bits y su respectiva descripción.

MuxR_In

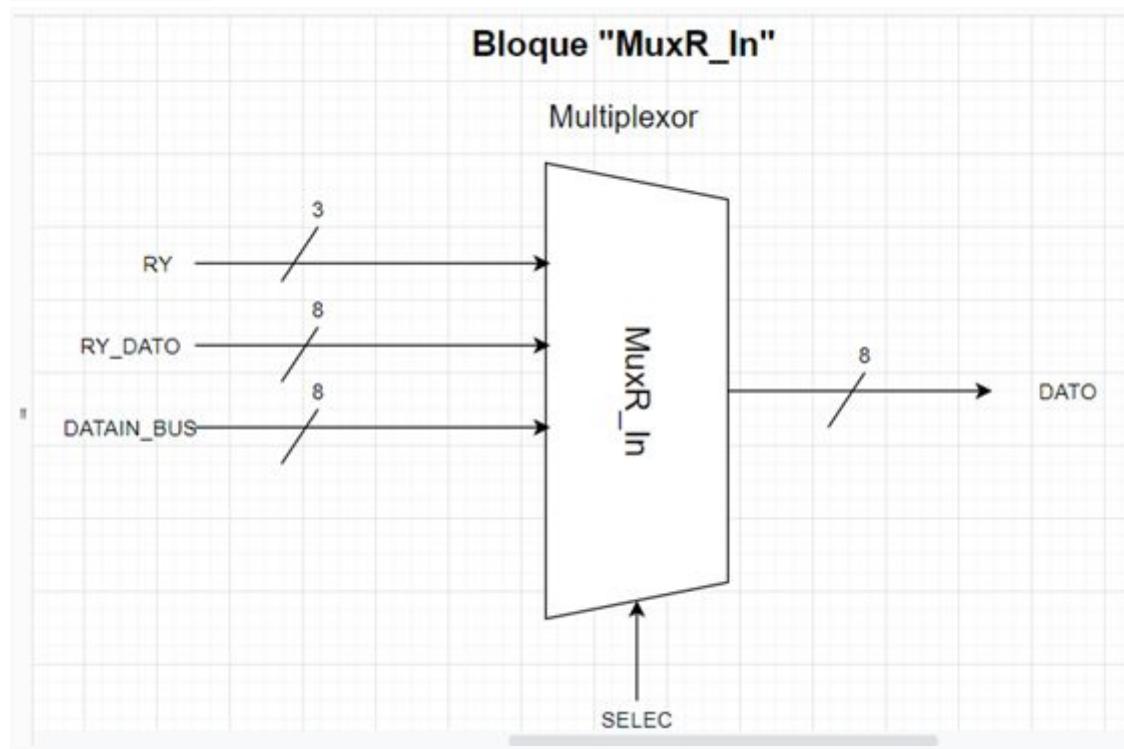
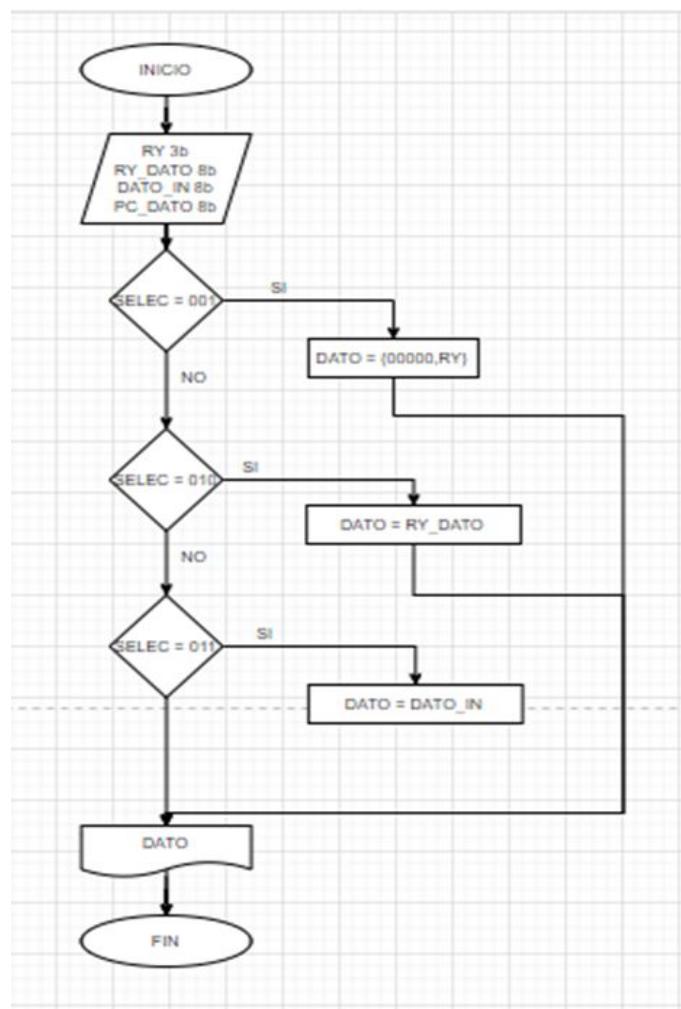


Figura 11.1. Diagrama del bloque MuxR_In.

señal	bits	descripcion
RY	3	Señal que almacena datos necesarios para realizar las operaciones
RY_DATO	8	Señal que toma el valor de RY en ciertas condiciones
DATAIN_BUS	8	Entrada de 8 bits que transmite la información a manipular.
DATO	8	Esta señal puede tener varios datos dependiendo del las entradas en el multiplexor.

Figura 11 .2. Tabla con los nombres de las señales del bloque, así como sus número de bits y su respectiva descripción.

Descripción funcional



ESQUEMÁTICOS

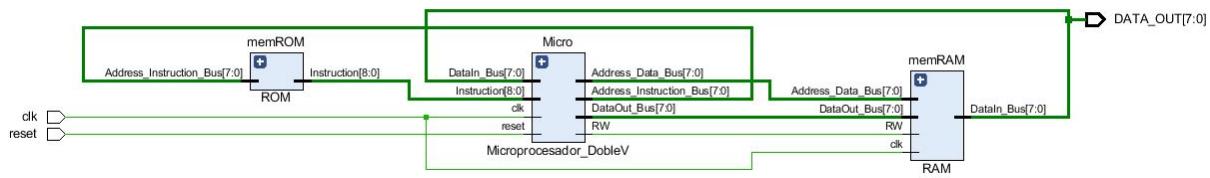


Figura 12.1. Esquemático final del controlador (incluye memorias).

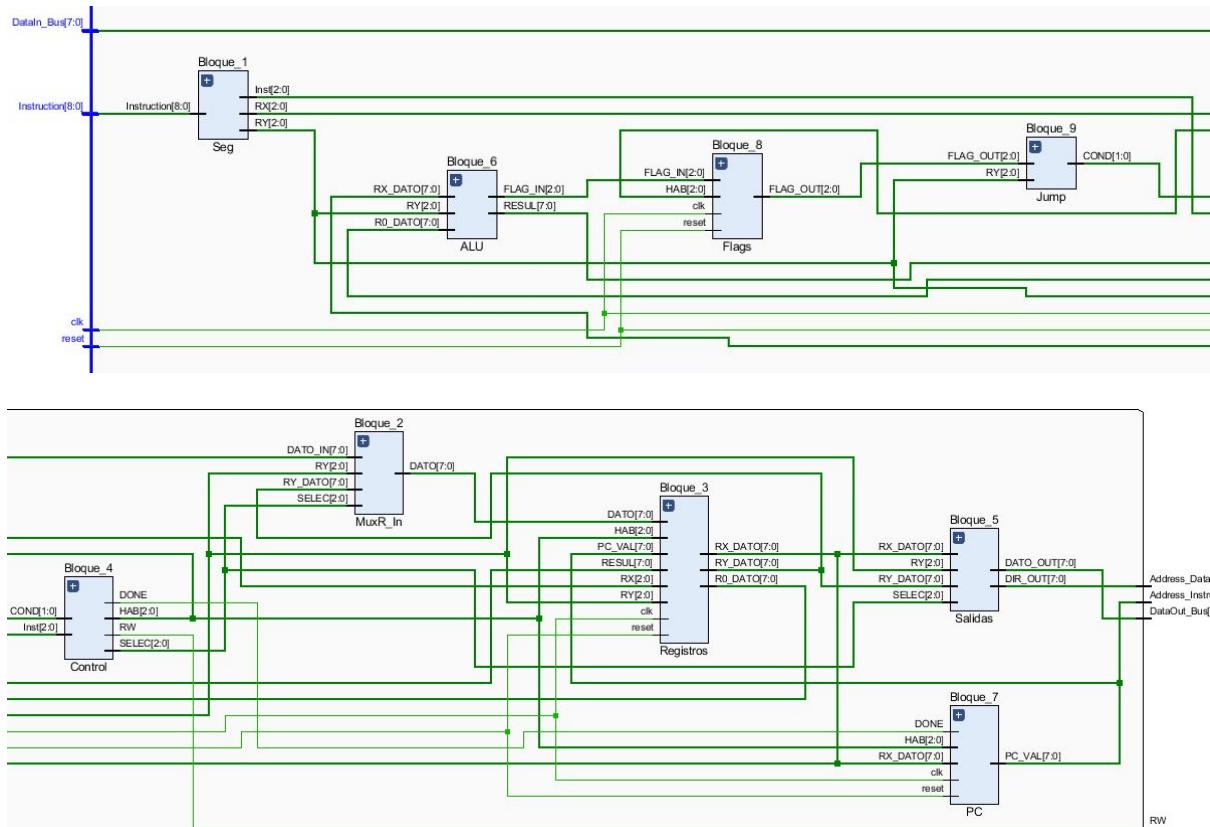


Figura 12.2. Esquemático final del microporcesador.

SIMULACIÓN

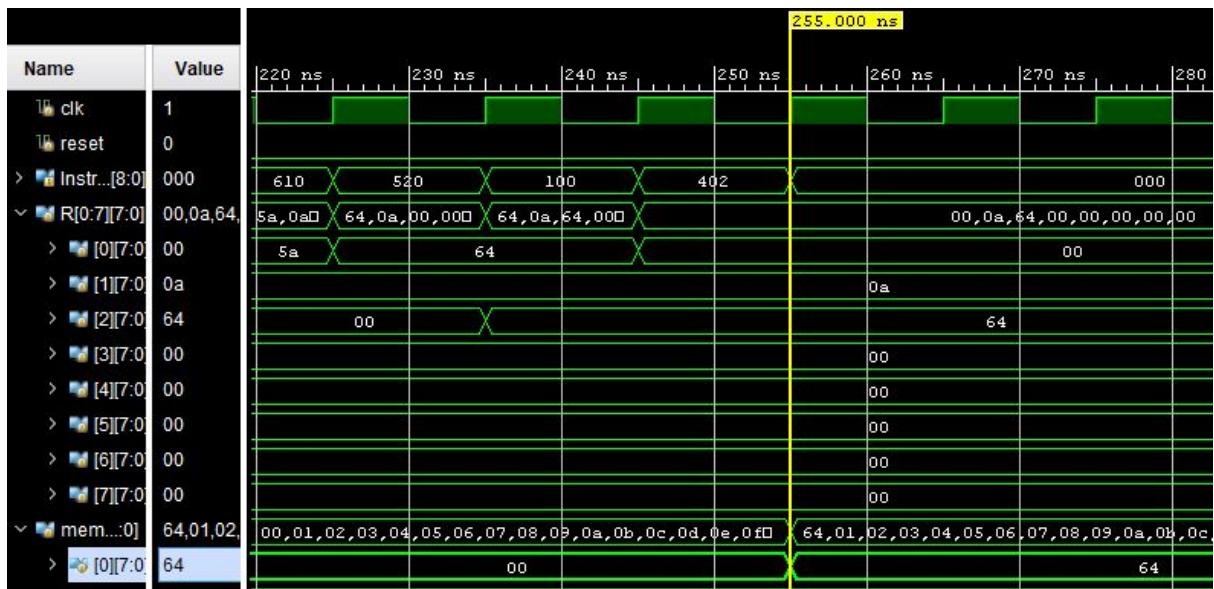


Figura 13.1. Simulación de la multiplicación de 10 por 10; el resultado se guarda en 00 de memRAM.

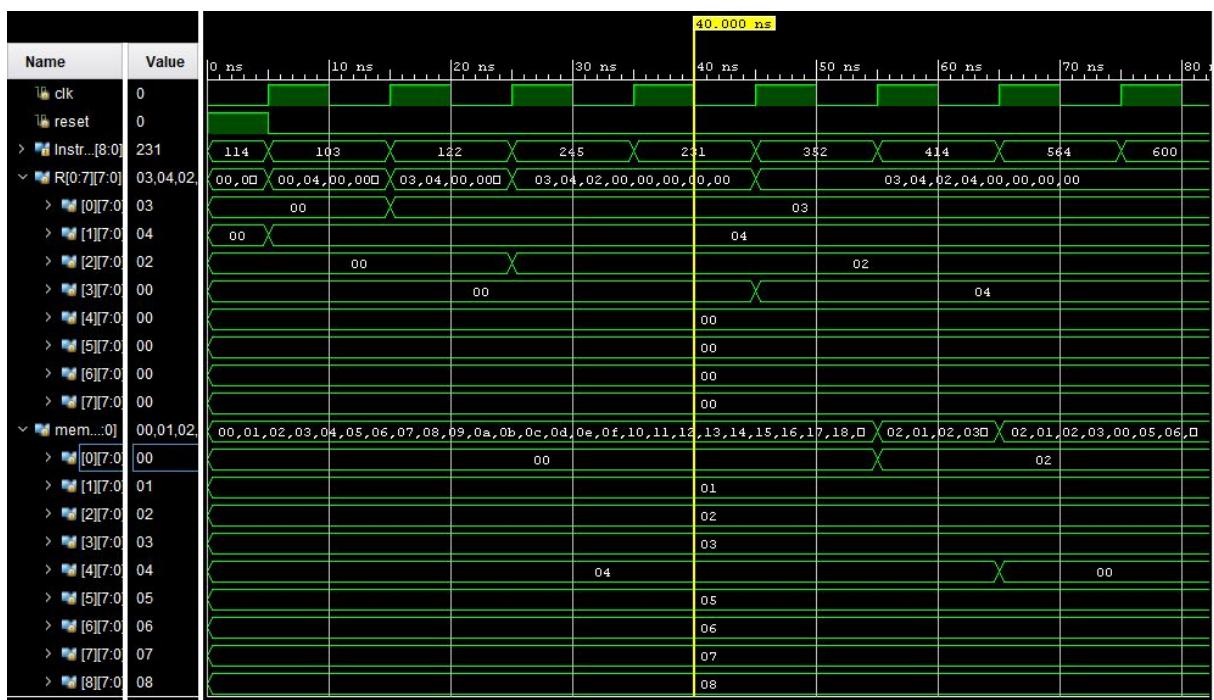


Figura 13.2. Simulación de cada instrucción.

Conclusiones

Los resultados finales de nuestro microprocesador fueron satisfactorios. Se logró cumplir con el set de instrucciones especificado y la realización de la multiplicación con el uso de un software. A pesar de los inconvenientes y problemas que se fueron presentando a lo largo de la elaboración, se llegó a solucionar estos detalles imprevistos y concluir con nuestro diseño.

Para concluir con este proyecto que se elaboró durante el curso de Microcontroladores, se nos explicó y tuvimos apoyo por parte de nuestro profesor para realizar cada parte a detalle.

El presente reporte muestra de manera clara y concisa que dicho comportamiento se obtiene. Dependiendo de los valores que se ingresan a la entrada, cada sección del programa se encarga de realizar la función descrita para que como resultado se obtenga un número.

A lo largo del proyecto, con la nueva modalidad del trabajo en equipo, la carga laboral disminuyó, ya que fue posible realizar diferentes tareas simultáneamente organizados en equipos, compartiendo en tiempo real cada una su parte, para que finalmente con una colaboración grupal obtener en conjunto una práctica en un tiempo menor, con muy buenos resultados.

Para llevar a cabo este proyecto debimos tener conocimientos de los sistemas digitales, software, hardware y sus funciones, además de cómo llevar a cabo cada una de ellas. También conocimientos nuevas funciones que la herramienta Vivado puede proporcionar..

APÉNDICE A (Código)

Registros.v

```
1. `timescale 1ns / 1ps
2.
3. module Registros(
4.     input reset,
5.     input clk,
6.     input [2:0] HAB,
7.     input [2:0] RX,
8.     input [2:0] RY,
9.     input [7:0] DATO,
10.    input [7:0] RESUL,
11.    input [7:0] PC_VAL,
12.    output [7:0] RY_DATO,
13.    output [7:0] RX_DATO,
14.    output [7:0] R0_DATO
15. );
16.
17. reg [7:0] R[0:7];
18.
19. assign R0_DATO = R[0];
20. assign RX_DATO = R[RX];
21. assign RY_DATO = R[RY];
22.
23. always@(posedge clk, posedge reset)
24. begin
25.     if(reset == 1)
26.         begin
27.             R[0] <= 8'b00000000;
28.             R[1] <= 8'b00000000;
29.             R[2] <= 8'b00000000;
30.             R[3] <= 8'b00000000;
31.             R[4] <= 8'b00000000;
32.             R[5] <= 8'b00000000;
33.             R[6] <= 8'b00000000;
34.             R[7] <= 8'b00000000;
35.         end
36.     else
37.         begin
38.             case(HAB)
39.                 3'b001: R[0] <= RESUL;
40.                 3'b011: R[7] <= PC_VAL;
41.                 3'b100: R[RX] <= DATO;
42.                 default:
43.                     begin
44.                         R[0] <= R[0];
45.                         R[1] <= R[1];
46.                         R[2] <= R[2];
47.                         R[3] <= R[3];
48.                         R[4] <= R[4];
49.                         R[5] <= R[5];
50.                         R[6] <= R[6];
51.                         R[7] <= R[7];
52.                     end
53.                 endcase
54.             end
55.         end
56.     end
57.endmodule
```

Control.v

```
1. `timescale 1ns / 1ps
2.
3. module Control(
4.     input [2:0] Inst,
5.     input [1:0] COND,
6.     output reg RW,
7.     output reg [2:0] SELEC,
8.     output reg DONE,
9.     output reg [2:0] HAB
10. );
11.
12.    always @*
13.    begin
14.        case(Inst)
15.            3'b001:
16.                begin
17.                    RW <= 1'b0;
18.                    SELEC <= 001;
19.                    DONE <= 1;
20.                    HAB <= 100;
21.                end
22.            3'b010:
23.                begin
24.                    RW <= 0;
25.                    SELEC <= 011;
26.                    DONE <= 1;
27.                    HAB <= 100;
28.                end
29.            3'b011:
30.                begin
31.                    RW <= 1;
32.                    SELEC <= 101;
33.                    DONE <= 1;
34.                    HAB <= 000;
35.                end
36.            3'b100:
37.                begin
38.                    RW <= 1;
39.                    SELEC <= 110;
40.                    DONE <= 1;
41.                    HAB <= 000;
42.                end
43.            3'b101:
44.                begin
45.                    RW <= 1'b0;
46.                    SELEC <= 010;
47.                    DONE <= 1;
48.                    HAB <= 100;
49.                end
50.            3'b110:
51.                begin
52.                    RW <= 1'b0;
53.                    SELEC <= 000;
54.                    DONE <= 1;
55.                    HAB <= 001;
56.                end
57.            3'b111:
```

```

58.      begin
59.          case (COND)
60.              2'b01:
61.                  begin
62.                      RW <= 1'b0;
63.                      SELEC <= 000;
64.                      DONE <= 1;
65.                      HAB <= 010;
66.                  end
67.              2'b11:
68.                  begin
69.                      RW <= 1'b0;
70.                      SELEC <= 000;
71.                      DONE <= 1;
72.                      HAB <= 011;
73.                  end
74.              default:
75.                  begin
76.                      RW <= 1'b0;
77.                      SELEC <= 000;
78.                      DONE <= 1;
79.                      HAB <= 000;
80.                  end
81.          endcase
82.      end
83.  default:
84.      begin
85.          RW <= 1'b0;
86.          SELEC <= 3'b000;
87.          DONE <= 1'b0;
88.          HAB <= 3'b000;
89.      end
90.  endcase
91. end
92.endmodule

```

Seg.v

```
1. `timescale 1ns / 1ps
2.
3. module Seg(
4.     input [8:0] Instruction,
5.     output [2:0] Inst,
6.     output [2:0] RX,
7.     output [2:0] RY
8. );
9.
10.    assign Inst = Instruction[8:6];
11.    assign RX = Instruction[5:3];
12.    assign RY = Instruction[2:0];
13.endmodule
```

PC.v

```
1. `timescale 1ns / 1ps
2.
3. module PC(
4.     input clk,
5.     input reset,
6.     input DONE,
7.     input [7:0] RX_DATO,
8.     input [2:0] HAB,
9.     output reg [7:0] PC_VAL
10. );
11.
12.always@(posedge clk, posedge reset)
13. begin
14.     if(reset == 1)
15.         PC_VAL <= 8'b00000000;
16.     else
17.         begin
18.             case(HAB)
19.                 3'b010: PC_VAL<=RX_DATO;
20.                 3'b011: PC_VAL<=RX_DATO;
21.             endcase
22.             if(DONE == 1'b1)
23.                 PC_VAL=PC_VAL+1;
24.         end
25.     end
26.endmodule
```

Salidas.v

```
1. `timescale 1ns / 1ps
2.
3. module Salidas(
4.     input [2:0] SELEC,
5.     input [7:0] RX_DATO,
6.     input [2:0] RY,
7.     input [7:0] RY_DATO,
8.     output reg [7:0] DATO_OUT,
```

```

9.      output reg [7:0] DIR_OUT
10.     );
11.
12.always@*
13.    begin
14.      case(SELEC)
15.        3'b011:
16.          begin
17.            DIR_OUT <= RY_DATO;
18.            DATO_OUT <= 8'b00000000;
19.          end
20.        3'b101:
21.          begin
22.            DIR_OUT <= RX_DATO;
23.            DATO_OUT <= {5'b00000, RY};
24.          end
25.        3'b110:
26.          begin
27.            DIR_OUT <= RX_DATO;
28.            DATO_OUT <= RY_DATO;
29.          end
30.        default:
31.          begin
32.            DIR_OUT <= 8'b00000000;
33.            DATO_OUT <= 8'b00000000;
34.          end
35.      endcase
36.    end
37.endmodule

```

ALU.v

```

1. `timescale 1ns / 1ps
2.
3. module ALU(
4.   input [7:0] RX_DATO,
5.   input [7:0] R0_DATO,
6.   input [2:0] RY,
7.   output reg [2:0] FLAG_IN,
8.   output [7:0] RESUL
9. );
10.
11. reg [8:0] VALOR;
12.
13. assign RESUL = VALOR[7:0];
14. always@*
15.   begin
16.     case(RY)
17.       3'b000: VALOR <= R0_DATO + RX_DATO;
18.       3'b001: VALOR <= R0_DATO - RX_DATO;
19.       3'b010: VALOR <= R0_DATO << RX_DATO;
20.       3'b011: VALOR <= R0_DATO >> RX_DATO;
21.       3'b100: VALOR <= ~RX_DATO;
22.       3'b101: VALOR <= R0_DATO & RX_DATO;
23.       3'b110: VALOR <= R0_DATO | RX_DATO;
24.       3'b111: VALOR <= R0_DATO ^ RX_DATO;
25.     default: VALOR<= 8'b00000000;
26.   endcase
27.   if (VALOR[8] == 1) //CARRY
28.     FLAG_IN[1] <= 1;
29.   else FLAG_IN[1] <= 0;
30.   if (VALOR == 9'b000000000) //ZERO

```

```

31.          FLAG_IN[2] <= 1;
32.      else FLAG_IN[2] <= 0;
33.      if (VALOR[7] == 1'b1)           //SIGN
34.          FLAG_IN[0] <= 1;
35.      else FLAG_IN[0] <= 0;
36.    end
37.endmodule

```

Flags.v

```

1. `timescale 1ns / 1ps
2. module Flags(
3.     input [2:0] FLAG_IN,
4.     input clk,
5.     input reset,
6.     input [2:0] HAB,
7.     output reg [2:0] FLAG_OUT
8. );
9. always@(posedge clk, posedge reset)
10. begin
11.     if (reset == 1)
12.         FLAG_OUT <= 3'b000;
13.     else
14.         if(HAB == 001)
15.             FLAG_OUT <= FLAG_IN;
16. end
17.endmodule

```

Jump.v

```

1. `timescale 1ns / 1ps
2.
3. module Jump(
4.     input [2:0] FLAG_OUT,
5.     input [2:0] RY,
6.     output reg[1:0] COND
7. );
8. always@(FLAG_OUT,RY)
9. begin
10.     case(RY)
11.         3'b000: COND <= 2'b01;
12.         3'b001: COND <= 2'b11;
13.         3'b010:
14.             begin
15.                 if (FLAG_OUT[2] == 1)
16.                     COND <= 2'b01;
17.                 else
18.                     COND <= 2'b00;
19.             end
20.         3'b011:
21.             begin
22.                 if (FLAG_OUT[2] == 0)
23.                     COND <= 2'b01;
24.                 else
25.                     COND <= 2'b00;
26.             end
27.         3'b100:

```

```

28.          begin
29.              if (FLAG_OUT[1] == 1)
30.                  COND <= 2'b01;
31.              else
32.                  COND <= 2'b00;
33.          end
34.      3'b101:
35.          begin
36.              if (FLAG_OUT[1] == 0)
37.                  COND <= 2'b01;
38.              else
39.                  COND <= 2'b00;
40.          end
41.      3'b110:
42.          begin
43.              if (FLAG_OUT[0] == 1)
44.                  COND <= 2'b01;
45.              else
46.                  COND <= 2'b00;
47.          end
48.      3'b111:
49.          begin
50.              if (FLAG_OUT[0] == 0)
51.                  COND <= 2'b01;
52.              else
53.                  COND <= 2'b00;
54.          end
55.      default:
56.          begin
57.              COND <= 2'b00;
58.          end
59.      endcase
60.  end
61.endmodule

```

MuxR_In.v

```

1. `timescale 1ns / 1ps
2.
3. module MuxR_In(
4.     input [2:0] RY,
5.     input [7:0] RY_DATO,
6.     input [7:0] DATO_IN,
7.     input [2:0] SELEC,
8.     output reg [7:0] DATO
9. );
10.
11.    always@* begin
12.        case(SELEC)
13.            3'b001: DATO <= {5'b00000,RY};
14.            3'b010: DATO <= RY_DATO;
15.            3'b011: DATO <= DATO_IN;
16.            default: DATO <= 8'b00000000;
17.    endcase
18. end
19.endmodule

```

APÉNDICE B (TestBench)

ALU_TB.v

```
1. `timescale 1ns / 1ps
2.
3. module ALU_TB(
4.     );
5.     reg [7:0] RX_DATO;
6.     reg [7:0] R0_DATO;
7.     reg [2:0] RY;
8.     wire [2:0] FLAG_IN;
9.     wire [7:0] RESUL;
10.
11.    ALU DUT (
12.        .RX_DATO(RX_DATO),
13.        .R0_DATO(R0_DATO),
14.        .RY(RY),
15.        .FLAG_IN(FLAG_IN),
16.        .RESUL(RESUL)
17.    );
18.    initial
19.        begin
20.            RY <= 3'b000; RX_DATO <= 8'b00000000; R0_DATO <=
21.                8'b00000000;
22.            #10 RY <= 3'b000; RX_DATO <= 8'b00000001; R0_DATO <=
23.                8'b00000011;
24.            #10 RY <= 3'b001; RX_DATO <= 8'b00000011; R0_DATO <=
25.                8'b00000011;
26.            #10 RY <= 3'b010; RX_DATO <= 8'b00000100; R0_DATO <=
27.                8'b00000001;
28.            #10 RY <= 3'b011; RX_DATO <= 8'b00101010; R0_DATO <=
29.                8'b00000111;
30.        end
30.endmodule
```

Control_TB.v

```
1. `timescale 1ns / 1ps
2.
3. module Control_TB(
4.     );
5.     reg [2:0] Inst;
6.     reg [1:0] COND;
7.     wire RW;
```

```

8.      wire [2:0] SELEC;
9.      wire DONE;
10.     wire [2:0] HAB;
11.
12.    Control DUT(
13.        .Inst(Inst),
14.        .COND(COND),
15.        .RW(RW),
16.        .SELEC(SELEC),
17.        .DONE(DONE),
18.        .HAB(HAB)
19.    );
20.
21.    initial
22.        begin
23.            Inst <= 3'b000; COND <= 2'b00;
24.            #10 Inst <= 3'b001; COND <= 2'b00;
25.            #10 Inst <= 3'b010; COND <= 2'b00;
26.            #10 Inst <= 3'b011; COND <= 2'b00;
27.            #10 Inst <= 3'B100; COND <= 2'b00;
28.            #10 Inst <= 3'b101; COND <= 2'b00;
29.            #10 Inst <= 3'b110; COND <= 2'b00;
30.
31.            #10 Inst <= 3'b111; COND <= 2'b00;
32.            #10 Inst <= 3'b111; COND <= 2'b01;
33.            #10 Inst <= 3'b111; COND <= 2'b11;
34.            #10 Inst <= 3'b000; COND <= 2'b00;
35.        end
36.endmodule

```

Flags_TB.v

```

1. `timescale 1ns / 1ps
2.
3. module Flags_TB(
4. );
5.
6.     reg [2:0] FLAG_IN;
7.     reg clk;
8.     reg reset;
9.     reg [2:0] HAB;
10.    wire[2:0] FLAG_OUT;
11.
12.    Flags DUT(
13.        .FLAG_IN(FLAG_IN),
14.        .clk(clk),
15.        .reset(reset),
16.        .HAB(HAB),
17.        .FLAG_OUT(FLAG_OUT)
18.    );
19.
20.    initial
21.        begin
22.            clk<=0;

```

```

23.          reset<=1;
24.          #10 reset <= 0;
25.          #10 FLAG_IN <= 3'b000; HAB <= 3'b000;
26.          #10 FLAG_IN <= 3'b111; HAB <= 3'b000;
27.          #10 FLAG_IN <= 3'b111; HAB <= 3'b001;
28.          #10 FLAG_IN <= 3'b100; HAB <= 3'b001;
29.          #10 FLAG_IN <= 3'b000; HAB <= 3'b000;
30.      end
31.  always@(clk)
32.    #5 clk <= ~clk;
33.endmodule

```

Jump_TB.v

```

1. `timescale 1ns / 1ps
2.
3. module Jump_TB(
4. );
5.
6.   reg [2:0] FLAG_OUT;
7.   reg [2:0] RY;
8.   wire [1:0] COND;
9.
10.  Jump DUT (
11.    .FLAG_OUT (FLAG_OUT),
12.    .RY (RY),
13.    .COND (COND)
14.  );
15.
16. initial
17. begin
18.   FLAG_OUT <= 3'b000; RY <= 3'b000;
19.   #10 FLAG_OUT <= 3'b000; RY <= 3'b001;
20.   #10 FLAG_OUT <= 3'b100; RY <= 3'b010;
21.   #10 FLAG_OUT <= 3'b001; RY <= 3'b011;
22.   #10 FLAG_OUT <= 3'b010; RY <= 3'b100;
23.   #10 FLAG_OUT <= 3'b101; RY <= 3'b101;
24.   #10 FLAG_OUT <= 3'b001; RY <= 3'b110;
25.   #10 FLAG_OUT <= 3'b000; RY <= 3'b111;
26.
27.   #10 FLAG_OUT <= ~3'b000; RY <= 3'b000;
28.   #10 FLAG_OUT <= ~3'b000; RY <= 3'b001;
29.   #10 FLAG_OUT <= ~3'b100; RY <= 3'b010;
30.   #10 FLAG_OUT <= ~3'b001; RY <= 3'b011;
31.   #10 FLAG_OUT <= ~3'b010; RY <= 3'b100;
32.   #10 FLAG_OUT <= ~3'b101; RY <= 3'b101;
33.   #10 FLAG_OUT <= ~3'b001; RY <= 3'b110;
34.   #10 FLAG_OUT <= ~3'b000; RY <= 3'b111;
35. end
36.endmodule

```

MuxR_In_TB.v

```
1. `timescale 1ns / 1ps
2.
3. module MuxR_In_TB(
4.     );
5.
6.     reg [2:0] RY;
7.     reg [7:0] RY_DATO;
8.     reg [7:0] DATO_IN;
9.     reg [2:0] SELEC;
10.    wire [7:0] DATO;
11.
12.    MuxR_In DUT (
13.        .RY(RY),
14.        .RY_DATO(RY_DATO),
15.        .DATO_IN(DATO_IN),
16.        .SELEC(SELEC),
17.        .DATO(DATO)
18.    );
19.
20.    initial
21.        begin
22.            RY <= 3'b000; RY_DATO <= 8'b00000000; DATO_IN <=
23.                8'b00000000; SELEC <= 3'b000;
24.                #10 RY <= 3'b101; RY_DATO <= 8'b10101010; DATO_IN <=
25.                    8'b00001111; SELEC <= 3'b001;
26.                #10 RY <= 3'b101; RY_DATO <= 8'b10101010; DATO_IN <=
27.                    8'b00001111; SELEC <= 3'b010;
28.                #10 RY <= 3'b101; RY_DATO <= 8'b10101010; DATO_IN <=
29.                    8'b00001111; SELEC <= 3'b011;
30.        end
31.    endmodule
```

PC_TB.v

```
1. `timescale 1ns / 1ps
2.
3. module PC_TB(
4.     );
5.     reg clk;
6.     reg reset;
7.     reg DONE;
8.     reg [7:0] RX_DATO;
9.     reg [2:0] HAB;
10.    wire[7:0] PC_VAL;
11.
12.    PC DUT (
13.        .clk(clk),
14.        .reset(reset),
15.        .DONE(DONE),
16.        .RX_DATO(RX_DATO),
17.        .HAB(HAB),
18.        .PC_VAL(PC_VAL)
19.    );
```

```

20.
21.     initial
22.         begin
23.             clk <= 0;
24.             reset<=1;
25.             DONE <= 1'b0; RX_DATO <= 8'b00000000; HAB <= 3'b000;
26.             #10 reset <= 0;
27.             //Incremento
28.             #10 DONE <= 1'b1; RX_DATO <= 8'b00000000; HAB <= 3'b000;
29.             #10 DONE <= 1'b1; RX_DATO <= 8'b00000000; HAB <= 3'b000;
30.             #10 DONE <= 1'b1; RX_DATO <= 8'b00000000; HAB <= 3'b000;
31.             #10 DONE <= 1'b1; RX_DATO <= 8'b00000000; HAB <= 3'b000;
32.
33.             //JUMP
34.             #10 DONE <= 1'b1; RX_DATO <= 8'b01010101; HAB <= 3'b010;
35.             #10 DONE <= 1'b1; RX_DATO <= 8'b11001100; HAB <= 3'b011;
36.             #10 DONE <= 1'b0; RX_DATO <= 8'b00000000; HAB <= 3'b000;
37.         end
38.     always@ (clk)
39.         #5 clk <= ~clk;
40.endmodule

```

Registros_TB.v

```

1. `timescale 1ns / 1ps
2.
3. module Registros_TB(
4. );
5.     reg reset;
6.     reg clk;
7.     reg [2:0] HAB;
8.     reg [2:0] RX;
9.     reg [2:0] RY;
10.    reg [7:0] DATO;
11.    reg [7:0] RESUL;
12.    reg [7:0] PC_VAL;
13.    wire [7:0] RY_DATO;
14.    wire [7:0] RX_DATO;
15.    wire [7:0] R0_DATO;
16.
17.    Registros DUT(
18.        .reset(reset),
19.        .clk(clk),
20.        .HAB(HAB),
21.        .RX(RX),
22.        .RY(RY),
23.        .DATO(DATO),
24.        .RESUL(RESUL),
25.        .PC_VAL(PC_VAL),
26.        .RY_DATO(RY_DATO),
27.        .RX_DATO(RX_DATO),
28.        .R0_DATO(R0_DATO)
29.    );
30.    initial

```

```

31.      begin
32.          clk<=0;
33.          reset<=1;
34.          HAB <= 3'b000; RX <= 3'b000; RY <= 3'b000; DATO <=
35.              8'b00000000; RESUL <= 8'b00000000; PC_VAL <= 8'b00000000;
36.              #20 reset<=0;
37.              #20 HAB <= 3'b100; RX <= 3'b111; RY <= 3'b100; DATO <=
38.                  8'b01010101; RESUL <= 8'b00000000; PC_VAL <= 8'b00000000;
39.                  #20 HAB <= 3'b100; RX <= 3'b001; RY <= 3'b110; DATO <=
40.                      8'b10101010; RESUL <= 8'b00000000; PC_VAL <= 8'b00000000;
41.                      #20 HAB <= 3'b100; RX <= 3'b100; RY <= 3'b111; DATO <=
42.                          8'b00110011; RESUL <= 8'b00000000; PC_VAL <= 8'b00000000;
43.                          #20 HAB <= 3'b001; RX <= 3'b011; RY <= 3'b100; DATO <=
44.                              8'b00000000; RESUL <= 8'b11110000; PC_VAL <= 8'b00000000;
45.                              #20 HAB <= 3'b011; RX <= 3'b011; RY <= 3'b100; DATO <=
46.                                  8'b00000000; RESUL <= 8'b00000000; PC_VAL <= 8'b00001111;
47.      end
48.
49.      always@(clk)
50.          #5 clk <= ~clk;
51.endmodule

```

Salidas_TB.v

```

1. `timescale 1ns / 1ps
2.
3. module Salidas_TB(
4.     );
5.     reg [2:0] SELEC;
6.     reg [7:0] RX_DATO;
7.     reg [2:0] RY;
8.     reg [7:0] RY_DATO;
9.     wire [7:0] DATO_OUT;
10.    wire [7:0] DIR_OUT;
11.
12.    Salidas DUT(
13.        .SELEC(SELEC),
14.        .RX_DATO(RX_DATO),
15.        .RY(RY),
16.        .RY_DATO(RY_DATO),
17.        .DATO_OUT(DATO_OUT),
18.        .DIR_OUT(DIR_OUT)
19.    );
20.    initial
21.        begin
22.            SELEC <= 3'b000; RX_DATO <= 8'b00000000; RY_DATO <=
23.                8'b00000000; RY <= 3'b000;
24.                #20 SELEC <= 3'b011; RX_DATO <= 8'b01010101; RY_DATO <=
25.                    8'b00110011; RY <= 3'b110;
26.                    #20 SELEC <= 3'b101; RX_DATO <= 8'b10010010; RY_DATO <=
27.                        8'b11100011; RY <= 3'b010;
28.                        #20 SELEC <= 3'b110; RX_DATO <= 8'b11001100; RY_DATO <=
29.                            8'b00011100; RY <= 3'b001;
30.                            end
31.
32.
33.
34.
35.
36.
37.
38.
39.
40.
41.
42.
43.
44.
45.
46.
47.
48.
49.
50.
51.
52.
53.
54.
55.
56.
57.
58.
59.
60.
61.
62.
63.
64.
65.
66.
67.
68.
69.
70.
71.
72.
73.
74.
75.
76.
77.
78.
79.
80.
81.
82.
83.
84.
85.
86.
87.
88.
89.
90.
91.
92.
93.
94.
95.
96.
97.
98.
99.
100.
101.
102.
103.
104.
105.
106.
107.
108.
109.
110.
111.
112.
113.
114.
115.
116.
117.
118.
119.
120.
121.
122.
123.
124.
125.
126.
127.
128.
129.
130.
131.
132.
133.
134.
135.
136.
137.
138.
139.
140.
141.
142.
143.
144.
145.
146.
147.
148.
149.
150.
151.
152.
153.
154.
155.
156.
157.
158.
159.
160.
161.
162.
163.
164.
165.
166.
167.
168.
169.
170.
171.
172.
173.
174.
175.
176.
177.
178.
179.
180.
181.
182.
183.
184.
185.
186.
187.
188.
189.
190.
191.
192.
193.
194.
195.
196.
197.
198.
199.
200.
201.
202.
203.
204.
205.
206.
207.
208.
209.
210.
211.
212.
213.
214.
215.
216.
217.
218.
219.
220.
221.
222.
223.
224.
225.
226.
227.
228.
229.
230.
231.
232.
233.
234.
235.
236.
237.
238.
239.
240.
241.
242.
243.
244.
245.
246.
247.
248.
249.
250.
251.
252.
253.
254.
255.
256.
257.
258.
259.
259.
260.
261.
262.
263.
264.
265.
266.
267.
268.
269.
269.
270.
271.
272.
273.
274.
275.
276.
277.
278.
279.
279.
280.
281.
282.
283.
284.
285.
286.
287.
288.
289.
289.
290.
291.
292.
293.
294.
295.
296.
297.
298.
299.
299.
300.
301.
302.
303.
304.
305.
306.
307.
308.
309.
309.
310.
311.
312.
313.
314.
315.
316.
317.
318.
319.
319.
320.
321.
322.
323.
324.
325.
326.
327.
328.
329.
329.
330.
331.
332.
333.
334.
335.
336.
337.
338.
339.
339.
340.
341.
342.
343.
344.
345.
346.
347.
348.
349.
349.
350.
351.
352.
353.
354.
355.
356.
357.
358.
359.
359.
360.
361.
362.
363.
364.
365.
366.
367.
368.
369.
369.
370.
371.
372.
373.
374.
375.
376.
377.
378.
379.
379.
380.
381.
382.
383.
384.
385.
386.
387.
388.
389.
389.
390.
391.
392.
393.
394.
395.
396.
397.
398.
399.
399.
400.
401.
402.
403.
404.
405.
406.
407.
408.
409.
409.
410.
411.
412.
413.
414.
415.
416.
417.
418.
419.
419.
420.
421.
422.
423.
424.
425.
426.
427.
428.
429.
429.
430.
431.
432.
433.
434.
435.
436.
437.
438.
439.
439.
440.
441.
442.
443.
444.
445.
446.
447.
448.
449.
449.
450.
451.
452.
453.
454.
455.
456.
457.
458.
459.
459.
460.
461.
462.
463.
464.
465.
466.
467.
468.
469.
469.
470.
471.
472.
473.
474.
475.
476.
477.
478.
479.
479.
480.
481.
482.
483.
484.
485.
486.
487.
488.
489.
489.
490.
491.
492.
493.
494.
495.
496.
497.
498.
499.
499.
500.
501.
502.
503.
504.
505.
506.
507.
508.
509.
509.
510.
511.
512.
513.
514.
515.
516.
517.
518.
519.
519.
520.
521.
522.
523.
524.
525.
526.
527.
528.
529.
529.
530.
531.
532.
533.
534.
535.
536.
537.
538.
539.
539.
540.
541.
542.
543.
544.
545.
546.
547.
548.
549.
549.
550.
551.
552.
553.
554.
555.
556.
557.
558.
559.
559.
560.
561.
562.
563.
564.
565.
566.
567.
568.
569.
569.
570.
571.
572.
573.
574.
575.
576.
577.
578.
579.
579.
580.
581.
582.
583.
584.
585.
586.
587.
588.
589.
589.
590.
591.
592.
593.
594.
595.
596.
597.
598.
599.
599.
600.
601.
602.
603.
604.
605.
606.
607.
608.
609.
609.
610.
611.
612.
613.
614.
615.
616.
617.
618.
619.
619.
620.
621.
622.
623.
624.
625.
626.
627.
628.
629.
629.
630.
631.
632.
633.
634.
635.
636.
637.
638.
639.
639.
640.
641.
642.
643.
644.
645.
646.
647.
648.
649.
649.
650.
651.
652.
653.
654.
655.
656.
657.
658.
659.
659.
660.
661.
662.
663.
664.
665.
666.
667.
668.
669.
669.
670.
671.
672.
673.
674.
675.
676.
677.
678.
679.
679.
680.
681.
682.
683.
684.
685.
686.
687.
688.
689.
689.
690.
691.
692.
693.
694.
695.
696.
697.
698.
698.
699.
700.
701.
702.
703.
704.
705.
706.
707.
708.
709.
709.
710.
711.
712.
713.
714.
715.
716.
717.
718.
719.
719.
720.
721.
722.
723.
724.
725.
726.
727.
728.
729.
729.
730.
731.
732.
733.
734.
735.
736.
737.
738.
739.
739.
740.
741.
742.
743.
744.
745.
746.
747.
748.
749.
749.
750.
751.
752.
753.
754.
755.
756.
757.
758.
759.
759.
760.
761.
762.
763.
764.
765.
766.
767.
768.
769.
769.
770.
771.
772.
773.
774.
775.
776.
777.
778.
779.
779.
780.
781.
782.
783.
784.
785.
786.
787.
788.
788.
789.
789.
790.
791.
792.
793.
794.
795.
796.
797.
798.
798.
799.
800.
801.
802.
803.
804.
805.
806.
807.
808.
809.
809.
810.
811.
812.
813.
814.
815.
816.
817.
818.
819.
819.
820.
821.
822.
823.
824.
825.
826.
827.
828.
829.
829.
830.
831.
832.
833.
834.
835.
836.
837.
838.
839.
839.
840.
841.
842.
843.
844.
845.
846.
847.
848.
849.
849.
850.
851.
852.
853.
854.
855.
856.
857.
858.
859.
859.
860.
861.
862.
863.
864.
865.
866.
867.
868.
869.
869.
870.
871.
872.
873.
874.
875.
876.
877.
878.
879.
879.
880.
881.
882.
883.
884.
885.
886.
887.
888.
889.
889.
890.
891.
892.
893.
894.
895.
896.
897.
898.
898.
899.
900.
901.
902.
903.
904.
905.
906.
907.
908.
909.
909.
910.
911.
912.
913.
914.
915.
916.
917.
918.
919.
919.
920.
921.
922.
923.
924.
925.
926.
927.
928.
929.
929.
930.
931.
932.
933.
934.
935.
936.
937.
938.
939.
939.
940.
941.
942.
943.
944.
945.
946.
947.
948.
948.
949.
950.
951.
952.
953.
954.
955.
956.
957.
958.
959.
959.
960.
961.
962.
963.
964.
965.
966.
967.
968.
969.
969.
970.
971.
972.
973.
974.
975.
976.
977.
978.
979.
979.
980.
981.
982.
983.
984.
985.
986.
987.
988.
989.
989.
990.
991.
992.
993.
994.
995.
996.
997.
998.
999.
999.
1000.
1000.
1001.
1002.
1003.
1004.
1005.
1006.
1007.
1008.
1009.
1009.
1010.
1011.
1012.
1013.
1014.
1015.
1016.
1017.
1018.
1019.
1019.
1020.
1021.
1022.
1023.
1024.
1025.
1026.
1027.
1028.
1029.
1029.
1030.
1031.
1032.
1033.
1034.
1035.
1036.
1037.
1038.
1039.
1039.
1040.
1041.
1042.
1043.
1044.
1045.
1046.
1047.
1048.
1049.
1049.
1050.
1051.
1052.
1053.
1054.
1055.
1056.
1057.
1058.
1059.
1059.
1060.
1061.
1062.
1063.
1064.
1065.
1066.
1067.
1068.
1069.
1069.
1070.
1071.
1072.
1073.
1074.
1075.
1076.
1077.
1078.
1079.
1079.
1080.
1081.
1082.
1083.
1084.
1085.
1086.
1087.
1088.
1088.
1089.
1090.
1091.
1092.
1093.
1094.
1095.
1096.
1097.
1097.
1098.
1099.
1099.
1100.
1101.
1102.
1103.
1104.
1105.
1106.
1107.
1108.
1109.
1109.
1110.
1111.
1112.
1113.
1114.
1115.
1116.
1117.
1118.
1119.
1119.
1120.
1121.
1122.
1123.
1124.
1125.
1126.
1127.
1128.
1129.
1129.
1130.
1131.
1132.
1133.
1134.
1135.
1136.
1137.
1138.
1139.
1139.
1140.
1141.
1142.
1143.
1144.
1145.
1146.
1147.
1148.
1148.
1149.
1150.
1151.
1152.
1153.
1154.
1155.
1156.
1157.
1158.
1159.
1159.
1160.
1161.
1162.
1163.
1164.
1165.
1166.
1167.
1168.
1169.
1169.
1170.
1171.
1172.
1173.
1174.
1175.
1176.
1177.
1178.
1178.
1179.
1180.
1181.
1182.
1183.
1184.
1185.
1186.
1187.
1187.
1188.
1189.
1190.
1191.
1192.
1193.
1194.
1195.
1195.
1196.
1197.
1198.
1199.
1199.
1200.
1201.
1202.
1203.
1204.
1205.
1206.
1207.
1208.
1209.
1209.
1210.
1211.
1212.
1213.
1214.
1215.
1216.
1217.
1218.
1219.
1219.
1220.
1221.
1222.
1223.
1224.
1225.
1226.
1227.
1228.
1229.
1229.
1230.
1231.
1232.
1233.
1234.
1235.
1236.
1237.
1238.
1239.
1239.
1240.
1241.
1242.
1243.
1244.
1245.
1246.
1247.
1248.
1248.
1249.
1250.
1251.
1252.
1253.
1254.
1255.
1256.
1257.
1258.
1259.
1259.
1260.
1261.
1262.
1263.
1264.
1265.
1266.
1267.
1268.
1269.
1269.
1270.
1271.
1272.
1273.
1274.
1275.
1276.
1277.
1278.
1278.
1279.
1280.
1281.
1282.
1283.
1284.
1285.
1286.
1287.
1287.
1288.
1289.
1290.
1291.
1292.
1293.
1294.
1295.
1296.
1297.
1297.
1298.
1299.
1299.
1300.
1301.
1302.
1303.
1304.
1305.
1306.
1307.
1308.
1309.
1309.
1310.
1311.
1312.
1313.
1314.
1315.
1316.
1317.
1318.
1319.
1319.
1320.
1321.
1322.
1323.
1324.
1325.
1326.
1327.
1328.
1329.
1329.
1330.
1331.
1332.
1333.
1334.
1335.
1336.
1337.
1338.
1339.
1339.
1340.
1341.
1342.
1343.
1344.
1345.
1346.
1347.
1348.
1348.
1349.
1350.
1351.
1352.
1353.
1354.
1355.
1356.
1357.
1358.
1359.
1359.
1360.
1361.
1362.
1363.
1364.
1365.
1366.
1367.
1368.
1369.
1369.
1370.
1371.
1372.
1373.
1374.
1375.
1376.
1377.
1378.
1378.
1379.
1380.
1381.
1382.
1383.
1384.
1385.
1386.
1387.
1387.
1388.
1389.
1390.
1391.
1392.
1393.
1394.
1395.
1396.
1397.
1397.
1398.
1399.
1399.
1400.
1401.
1402.
1403.
1404.
1405.
1406.
1407.
1408.
1408.
1409.
1410.
1411.
1412.
1413.
1414.
1415.
1416.
1417.
1418.
1418.
1419.
1420.
1421.
1422.
1423.
1424.
1425.
1426.
1427.
1428.
1428.
1429.
1430.
1431.
1432.
1433.
1434.
1435.
1436.
1437.
1438.
1438.
1439.
1440.
1441.
1442.
1443.
1444.
1445.
1446.
1447.
1448.
1448.
1449.
1450.
1451.
1452.
1453.
1454.
1455.
1456.
1457.
1458.
1459.
1459.
1460.
1461.
1462.
1463.
1464.
1465.
1466.
1467.
1468.
1468.
1469.
1470.
1471.
1472.
1473.
1474.
1475.
1476.
1477.
1478.
1478.
1479.
1480.
1481.
1482.
1483.
1484.
1485.
1486.
1487.
1487.
1488.
1489.
1490.
1491.
1492.
1493.
1494.
1495.
1496.
1497.
1498.
1498.
1499.
1500.
1501.
1502.
1503.
1504.
1505.
1506.
1507.
1508.
1509.
1509.
1510.
1511.
1512.
1513.
1514.
1515.
1516.
1517.
1518.
1518.
1519.
1520.
1521.
1522.
1523.
1524.
1525.
1526.
1527.
1528.
1529.
1529.
1530.
1531.
1532.
1533.
1534.
1535.
1536.
1537.
1538.
1538.
1539.
1540.
1541.
1542.
1543.
1544.
1545.
1546.
1547.
1548.
1548.
1549.
1550.
1551.
1552.
1553.
1554.
1555.
1556.
1557.
1558.
1559.
1559.
1560.
1561.
1562.
1563.
1564.
1565.
1566.
1567.
1568.
1568.
1569.
1570.
1571.
1572.
1573.
1574.
1575.
1576.
1577.
1578.
1578.
1579.
1580.
1581.
1582.
1583.
1584.
1585.
1586.
1587.
1587.
1588.
1589.
1590.
1591.
1592.
1593.
1594.
1595.
1596.
1597.
1598.
1599.
1599.
1600.
1601.
1602.
1603.
1604.
1605.
1606.
1607.
1608.
1609.
1609.
1610.
1611.
1612.
1613.
1614.
1615.
1616.
1617.
1618.
1618.
1619.
1620.
1621.
1622.
1623.
1624.
1625.
1626.
1627.
1628.
1628.
1629.
1630.
1631.
1632.
1633.
1634.
1635.
1636.
1637.
1638.
1638.
1639.
1640.
1641.
1642.
1643.
1644.
1645.
1646.
1647.
1648.
1648.
1649.
1650.
1651.
1652.
1653.
1654.
1655.
1656.
1657.
1658.
1659.
1659.
1660.
1661.
1662.
1663.
1664.
1665.
1666.
1667.
1668.
1668.
1669.
1670.
1671.
1672.
1673.
1674.
1675.
1676.
1677.
1678.
1678.
1679.
1680.
1681.
1682.
1683.
1684.
1685.
1686.
1687.
1687.
1688.
1689.
1690.
1691.
1692.
1693.
1694.
1695.
1696.
1697.
1698.
1698.
1699.
1700.
1701.
1702.
1703.
1704.
1705.
1706.
1707.
1708.
1709.
1709.
1710.
1711.
1712.
1713.
1714.
1715.
1716.
1717.
1718.
1718.
1719.
1720.
1721.
1722.
1723.
1724.
1725.
1726.
1727.
1728.
1728.
1729.
1730.
1731.
1732.
1733.
1734.
1735.
1736.
1737.
1738.
1738.
1739.
1740.
1741.
1742.
1743.
1744.
1745.
1746.
1747.
1748.
1748.
1749.
1750.
1751.
1752.
1753.
1754.
1755.
1756.
1757.
1758.
1759.
1759.
1760.
1761.
1762.
1763.
1764.
1765.
1766.
1767.
1768.
1768.
1769.
1770.
1771.
1772.
1773.
1774.
1775.
1776.
1777.
1778.
1778.
1779.
1780.
1781.
1782.
1783.
1784.
1785.
1786.
1787.
1787.
1788.
1789.
1790.
1791.
1792.
1793.
1794.
1795.
1796.
1797.
1798.
1798.
1799.
1800.
1801.
1802.
1803.
1804.
1805.
1806.
1807.
1808.
1809.
1809.
1810.
1811.
1812.
1813.
1814.
1815.
1816.
1817.
1818.
1818.
1819.
1820.
1821.
1822.
1823.
1824.
1825.
1826.
1827.
1828.
1828.
1829.
1830.
1831.
1832.
1833.
1834.
1835.
1836.
1837.
1838.
1838.
1839.
1840.
1841.
1842.
1843.
1844.
1845.
1846.
1847.
1848.
1848.
1849.
1850.
1851.
1852.
1853.
1854.
1855.
1856.
1857.
1858.
1859.
1859.
1860.
1861.
1862.
1863.
1864.
1865.
1866.
1867.
1868.
1868.
1869.
1870.
1871.
1872.
1873.
1874.
1875.
1876.
1877.
1878.
1878.
1879.
1880.
1881.
1882.
1883.
1884.
1885.
1886.
1887.
1888.
1888.
1889.
1890.
1891.
1892.
1893.
1894.
1895.
1896.
1897.
1898.
1898.
1899.
1900.
1901.
1902.
1903.
1904.
1905.
1906.
1907.
1908.
1909.
1909.
1910.
1911.
1912.
1913.
1914.
1915.
1916.
1917.
1918.
1918.
1919.
1920.
1921.
1922.
1923.
1924.
1925.
1926.
1927.
1928.
1929.
1929.
1930.
1931.
1932.
1933.
1934.
1935.
1936.
1937.
1938.
1938.
1939.
1940.
1941.
1942.
1943.
1944.
1945.
1946.
1947.
1948.
1948.
1949.
1950.
1951.
1952.
1953.
1954.
1955.
1956.
1957.
1958.
1959.
1959.
1960.
1961.
1962.
1963.
1964.
1965.
1966.
1967.
1968.
1968.
1969.
1970.
1971.
1972.
1973.
1974.
1975.
1976.
1977.
1978.
1978.
1979.
1980.
1981.
1982.
1983.
1984.
1985.
1986.
1987.
1988.
1988.
1989.
1990.
1991.
1992.
1993.
1994.
1995.
1996.
1997.
1998.
1999.
1999.
2000.
2001.
2002.
2003.
2004.
2005.
2006.
2007.
2008.
2009.
2009.
2010.
2011.
2012.
2013.
2014.
2015.
2016.
2017.
2018.
2019.
2019.
2020.
2021.
2022.
2023.
2024.
2025.
2026.
2027.
2028.
2029.
2029.
2030.
2031.
2032.
2033.
2034.
2035.
2036.
2037.
2038.
2038.
2039.
2040.
2041.
2042.
2043.
2044.
2
```

```
27.endmodule
```

Seq_TB.v

```
1. `timescale 1ns / 1ps
2.
3. module Seq_TB(
4. );
5.
6.     reg [8:0] Instruction;
7.     wire [2:0] Inst;
8.     wire [2:0] RX;
9.     wire [2:0] RY;
10.
11.    Seg DUT(
12.        .Instruction(Instruction),
13.        .Inst(Inst),
14.        .RX(RX),
15.        .RY(RY)
16.    );
17.
18.    initial
19.        begin
20.            Instruction <= 9'b0000000000;
21.            #10 Instruction <= 9'b001001001;
22.            #10 Instruction <= 9'b110101110;
23.            #10 Instruction <= 9'b011001100;
24.            #10 Instruction <= 9'b111111111;
25.            #10 Instruction <= 9'b0000000000;
26.        end
27.endmodule
```

RAM

```
1. `timescale 1ns / 1ps
2.
3. module RAM(
4.     input clk,
5.     input RW,
6.     input [7:0] Address_Data_Bus,
7.     input [7:0] DataOut_Bus,
8.     output[7:0] DataIn_Bus
9. );
10.
11.    reg [7:0] memRAM[0:255];
12.    assign DataIn_Bus = memRAM[Address_Data_Bus];
13.    initial
14.        $readmemh("RAM.mem",memRAM);
15.        always @(posedge clk)
16.            begin
17.                if(RW == 1)
18.                    memRAM[Address_Data_Bus] <= DataOut_Bus;
19.            end
20.endmodule
```

ROM

```
1. `timescale 1ns / 1ps
2.
3. module ROM(
4.     input [7:0] Address_Instruction_Bus,
5.     output[8:0] Instruction
6. );
7.
8.     reg [8:0] memROM[0:255];
9.     assign Instruction = memROM[Address_Instruction_Bus];
10.    initial
11.        $readmemb("ROM.mem",memROM);
12.endmodule
```

ROM_generator

Este programa ha sido elaborado en Python para introducir las instrucciones necesarias para realizar una multiplicación

```
1. file=open("D:/MicroUAZ_DobleV/MicroUAZ_DobleV.srcts/sources_1/new/ROM.mem",
2.           "w")
3.
4. x = int(input("Introduce el primer operando: "))
5. y = int(input("Introduce el segundo operando: "))
6.
7. file.write("001001001" + '\n')
8. for i in range(x):
9.     file.write("110001000" + '\n')
10. file.write("101001000" + '\n')
11. file.write("001000000" + '\n')
12.
13. for i in range(y):
14.     file.write("110001000" + '\n')
15. file.write("101010000" + '\n')
16. file.write("001000000" + '\n')
17.
18. file.write("100000010" + '\n')
19. file.write("000000000" + '\n')
20. z = x * y
21. print("Resultado en hex: " + hex(z))
22. print("Resultado en dec: " + str(z))
```