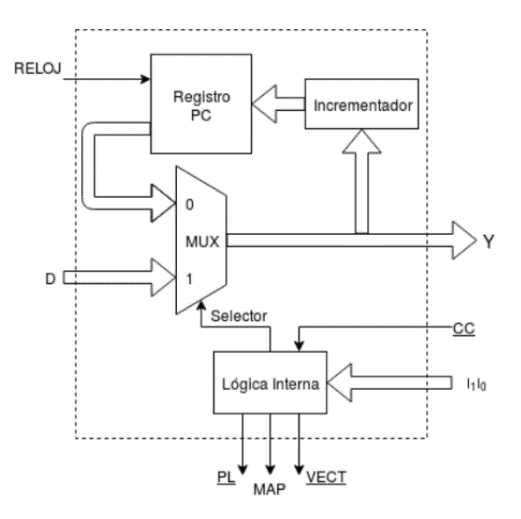
# **Objetivo**

Familiarizar al alumno en el conocimiento del secuenciador básico, el cual es una parte fundamental del procesador.

# **Introducción**

## Secuenciador básico

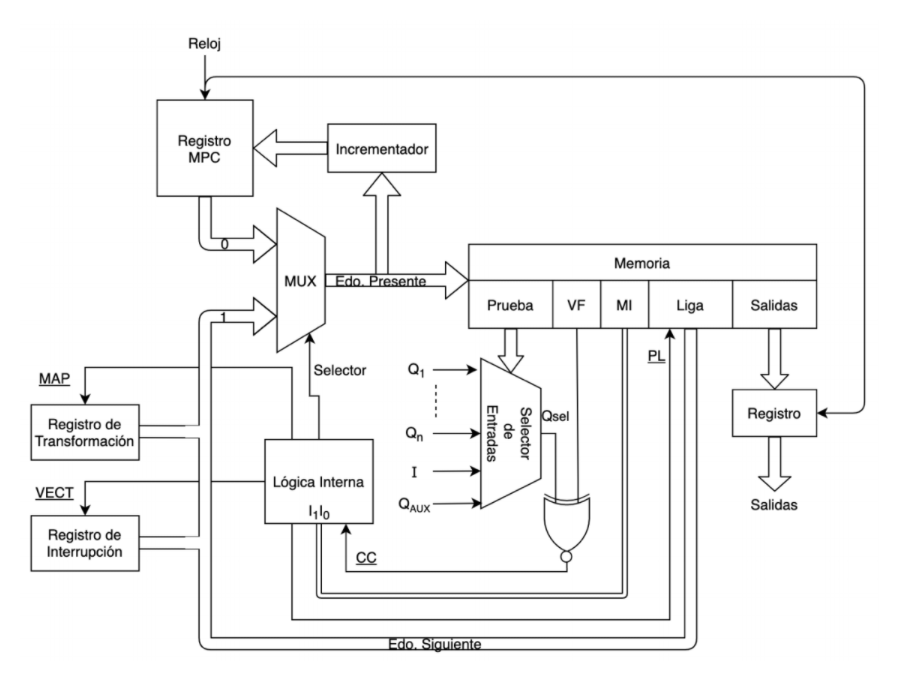
Para el diseño de los módulos de control de una computadora se requieren máquinas de estados que sean capaces de ejecutar algoritmos más complejos. Haciendo modificaciones y agregando componentes a la variante del direccionamiento implícito se pueden crear máquinas de estados que efectúen cartas ASM con llamadas a subrutinas, estructuras DO WHILE, iteraciones tipo FOR, entre otras. Los dispositivos que son capaces de efectuar este tipo de operaciones son llamados secuenciadores.

A continuación, se muestra el diagrama de bloques de un secuenciador básico. Como puede observar en el diagrama, la dirección del estado siguiente, dada por el bus Y, puede venir de dos lugares posibles: Del registro mPC o de la entrada D.

*Figura 1 Diagrama de bloques interno de un secuenciador básico*

El secuenciador cuenta con una lógica interna que se encarga de generar las señales que controlan al multiplexor. Dependiendo de la instrucción dada por las líneas I1 e I0 y de la línea CC, la lógica es capaz de seleccionar entre la salida del registro mPC o la entrada D.

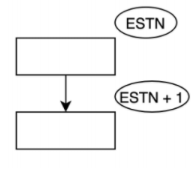
La lógica interna también genera las líneas PL, MAP y VECT, las cuales seleccionan unos registros cuyas salidas están conectadas a la entrada D del secuenciador. De esta forma la dirección de salto puede venir de tres lugares distintos.



*Figura 2 Diagrama del secuenciador básico conectado a una memora*

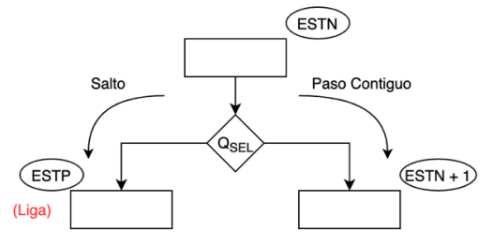
A continuación, se muestran las instrucciones que el secuenciador puede ejecutar y su representación en carta ASM.

## Paso Contiguo (C) [00]

En la instrucción continúa la dirección del estado siguiente la proporciona el registro mPC.

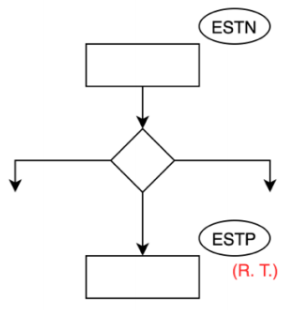
*Figura 3 Carta ASM de la microinstrucción Paso Contiguo.*

## Salto Condicional (SCO) [01]

En esta instrucción se revisa el valor de la línea CC, si es igual a uno, la dirección del estado siguiente la proporciona el registro mPC; si es igual a cero, la dirección del estado siguiente, contenida en el registro seleccionado por PL, ingresa a través de la entrada D.

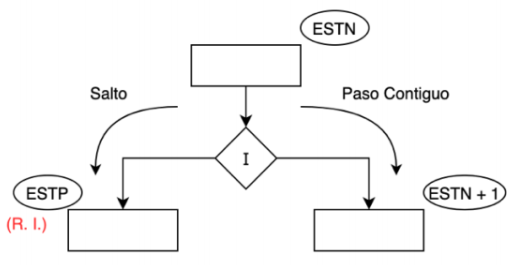
*Figura 4 Carta ASM de la microinstrucción Salto Condicional.*

## Salto de Transformación (ST) [10]

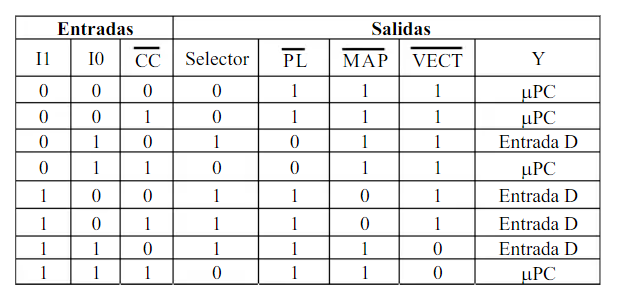
La dirección del estado siguiente se obtiene del registro seleccionado por la línea de MAP. Este registro también está conectado a la entrada D. Aquí se introduce una nueva notación de carta ASM: un rombo con varias bifurcaciones. La bifurcación que se elija dependerá del contenido del registro seleccionado por MAP.

*Figura 5 Carta ASM de la microinstrucción Salto de Transformación.*

## Salto de Interrupción (SCI) [11]

En esta instrucción se revisa el valor de CC, si es igual a uno, la dirección del estado siguiente proviene del registro mPC; si es igual a cero, la dirección del estado siguiente, contenida en el registro seleccionado por VECT, ingresa a través de la entrada D.

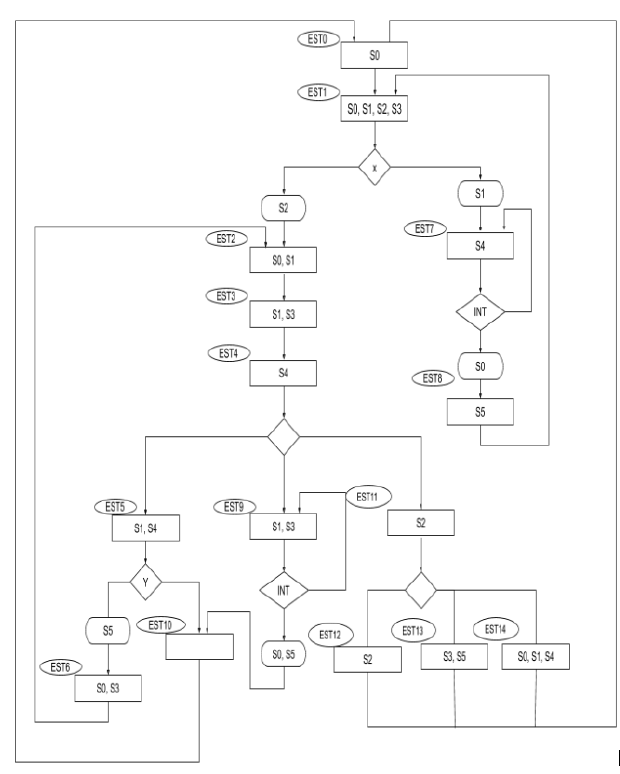
*Figura 6 Carta ASM de la microinstrucción Salto de Interrupción*

La lógica interna del secuenciador se construye a partir de la siguiente tabla poniendo énfasis en Y la cual nos indica de que bloque se tomara el estado siguiente.

# **Desarrollo**

Como primer punto, analizamos la carta ASM que se muestra a continuación, otorgando los valores binarios para cada uno de los estados.

*Figura 4 Carta ASM*



*Figura 7 Carta ASM a desarrollar*

1

1

1

1

0

0

0

0

*Figura 1.0 Carta ASM.*

Analizando primeramente la carta ASM, asegurándonos de que cumple con la regla “N, N+1, P”, realizamos la tabla de verdad correspondiente al comportamiento que esta tiene. Asignando inicialmente los valores binarios para cada uno de los estados y de los valores de prueba para identificarlos de manera correcta





PC – Paso Contiguo

SC – Salto Condicional

ST – Salto de Transformación

SI – Salto de Interrupción

*Figura 7.1 Valores binarios de los estados y las entradas.*

Con lo cual obtuvimos la siguiente tabla de verdad

*Figura 7.2 Tabla de verdad de la carta ASM.*

Con los valores obtenidos, implementamos la memoria en Quartus con la programación VHDL obteniendo el siguiente código:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY rom IS

   PORT (

       addr : IN std\_logic\_vector(3 DOWNTO 0);

       prueba : OUT std\_logic\_vector(1 DOWNTO 0);

       vf : OUT std\_logic;

       mi : OUT std\_logic\_vector(1 DOWNTO 0);

       liga : OUT std\_logic\_vector(3 DOWNTO 0);

       salida\_incremento : OUT std\_logic\_vector(5 DOWNTO 0);

       salida\_carga : OUT std\_logic\_vector(5 DOWNTO 0)

   );

END rom;

ARCHITECTURE behavioral OF rom IS

   TYPE mem\_rom IS ARRAY(0 TO 14) OF std\_logic\_vector(20 DOWNTO 0);

   SIGNAL data\_out : mem\_rom;

   SIGNAL data : std\_logic\_vector(20 DOWNTO 0);

BEGIN

   data\_out(0) <= "110000001000001000001";

   data\_out(1) <= "000010111001111001111";

   data\_out(2) <= "110000011000011000011";

   data\_out(3) <= "110000100001010001010";

   data\_out(4) <= "110100000010000010000";

   data\_out(5) <= "010011010110010010010";

   data\_out(6) <= "110010010001001001001";

   data\_out(7) <= "100110111010001010000";

   data\_out(8) <= "110010001100000100000";

   data\_out(9) <= "100111001101011001010";

   data\_out(10) <= "110010000000000000000";

   data\_out(11) <= "110100000000100000100";

   data\_out(12) <= "110010000000100000100";

   data\_out(13) <= "110010000101000101000";

   data\_out(14) <= "110010000010011010011";

   PROCESS (addr) BEGIN

       data <= data\_out(conv\_integer(unsigned(addr)));

       prueba <= data(20 DOWNTO 19);

       vf <= data(18);

       mi <= data(17 DOWNTO 16);

       liga <= data(15 DOWNTO 12);

       salida\_incremento <= data(11 DOWNTO 6);

       salida\_carga <= data(5 DOWNTO 0);

   END PROCESS;

END behavioral;

*Figura 7.3 Código rom.vhd*

Posteriormente implementamos la lógica del secuenciador como se describió en la introducción, teniendo como salidas a MAP, PL, VECT y el selector que nos sirve para saber si se esta haciendo un paso contiguo o un salto.

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY secuenciador\_logica IS

   PORT (

       mi : IN std\_logic\_vector(1 DOWNTO 0);

       ccn : IN std\_logic;

       mapn : OUT std\_logic;

       pln : OUT std\_logic;

       selector: OUT std\_logic;

       vectn : OUT std\_logic

   );

END secuenciador\_logica;

ARCHITECTURE behavioral OF secuenciador\_logica IS

    BEGIN

   PROCESS (mi, ccn) BEGIN

           mapn <= '0';

           pln <= '0';

           vectn <= '0';

           CASE mi IS

               WHEN "00" =>

                   selector <= '0';

               WHEN "01" =>

                   IF ccn = '1' THEN

                       selector <= '0';

                   ELSE

                       selector <= '1';

                       pln <= '1';

                   END IF;

               WHEN "10" =>

                   selector <= '0';

                   mapn <= '1';

               WHEN OTHERS =>

                   IF ccn = '1' THEN

                       selector <= '0';

                   ELSE

                       selector <= '1';

                       vectn <= '1';

                   END IF;

           END CASE;

   END PROCESS;

END behavioral;

*Figura 7.4 Código secuenciador\_logica.vhd*

Después, implementamos el selector de entrada con el cual conoceremos el valor de prueba que se esta utilizando conforme a lo descrito en la tabla de verdad.

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.STD\_LOGIC\_ARITH.ALL;

USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;

ENTITY selector\_entrada IS PORT (

   prueba : IN std\_logic\_vector (1 DOWNTO 0);

   X : IN std\_logic;

   Y : IN std\_logic;

   INT : IN std\_logic;

   AUX : IN std\_logic;

   valor\_entrada : OUT std\_logic);

END selector\_entrada;

ARCHITECTURE Behavioral OF selector\_entrada IS

BEGIN

   PROCESS (prueba) BEGIN

       CASE(prueba) IS

           WHEN "00" =>

           valor\_entrada <= X;

           WHEN "01" =>

           valor\_entrada <= Y;

           WHEN "10" =>

           valor\_entrada <= INT;

           WHEN OTHERS =>

           valor\_entrada <= AUX;

       END CASE;

   END PROCESS;

END Behavioral;

*Figura 7.5 Código selector\_entrada.vhd*

Así mismo creamos un selector de salida y un selector de liga para poder reconocer si se está obteniendo la salida de incremento o la salida de carga.

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.STD\_LOGIC\_ARITH.ALL;

USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;

ENTITY selector\_salida IS PORT (

   ccn : IN std\_logic;

   salida\_incremento : IN std\_logic\_vector (5 DOWNTO 0);

   salida\_carga : IN std\_logic\_vector (5 DOWNTO 0);

   salida : OUT std\_logic\_vector (5 DOWNTO 0));

END selector\_salida;

ARCHITECTURE Behavioral OF selector\_salida IS

BEGIN

   PROCESS (salida\_carga, salida\_incremento) BEGIN

       IF ccn = '1' THEN

           salida <= salida\_carga;

       ELSE

           salida <= salida\_incremento;

       END IF;

   END PROCESS;

END Behavioral;

*Figura 7.6 Código selector\_salida.vhd*

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.STD\_LOGIC\_ARITH.ALL;

USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;

ENTITY selector\_liga IS PORT (

   selector : IN std\_logic;

   liga\_incremento : IN std\_logic\_vector(3 DOWNTO 0);

   liga\_carga : IN std\_logic\_vector(3 DOWNTO 0);

   liga\_presente : OUT std\_logic\_vector(3 DOWNTO 0)

);

END selector\_liga;

ARCHITECTURE Behavioral OF selector\_liga IS

BEGIN

   PROCESS (selector) BEGIN

       IF selector = '1' THEN

           liga\_presente <= liga\_carga;

       ELSE

           liga\_presente <= liga\_incremento;

       END IF;

   END PROCESS;

END Behavioral;

*Figura 7.7 Código selector\_liga.vhd*

Y como se ha mencionado a lo largo de la introducción, realizamos el incrementador, que nos ayudara a pasar a un estado contiguo.

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.STD\_LOGIC\_ARITH.ALL;

USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;

ENTITY incremento IS PORT (

   liga\_presente : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

   liga\_incremento : OUT STD\_LOGIC\_VECTOR (3 DOWNTO 0)

);

END incremento;

ARCHITECTURE Behavioral OF incremento IS

BEGIN

   PROCESS(liga\_presente) BEGIN

       liga\_incremento <= liga\_presente + 1;

   END PROCESS;

END Behavioral;

*Figura 7.8 Código incremento.vhd*

Por último, implementamos una serie de registros de diferentes tamaños que nos sirve para poder manejar las salidas del secuenciador, uno para PL, MAP y VECT (registro\_tri.vhd); otro para MI (registro\_2.vhd); para CC (registro\_1.vhd) y finalmente para poder manipular la liga de incremento (registro\_4.vhd).

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY registro\_tri IS

   PORT (

       entrada : IN std\_logic\_vector(3 DOWNTO 0);

       activado : IN std\_logic;

         reset: IN std\_logic;

       clk : IN std\_logic;

         salida : OUT std\_logic\_vector(3 DOWNTO 0)

   );

END registro\_tri;

ARCHITECTURE behavioral OF registro\_tri IS

BEGIN

   PROCESS (clk, entrada, activado) BEGIN

       IF rising\_edge(clk) THEN

           IF activado = '0' or reset = '1' THEN

               salida <= "ZZZZ";

           ELSE

               salida <= entrada;

           END IF;

       END IF;

   END PROCESS;

END behavioral;

*Figura 7.9 Código registro\_tri.vhd*

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY registro\_1 IS

   PORT (

       entrada : IN std\_logic;

       reset : IN std\_logic;

       clk : IN std\_logic;

       salida : OUT std\_logic

   );

END registro\_1;

ARCHITECTURE behavioral OF registro\_1 IS

SIGNAL reg : std\_logic;

BEGIN

   PROCESS (clk, entrada) BEGIN

       IF rising\_edge(clk) THEN

           IF reset = '1' THEN

               reg <= 'Z';

           ELSE

               reg <= entrada;

           END IF;

       END IF;

       salida <= reg;

   END PROCESS;

END behavioral;

*Figura 7.10 Código registro\_1.vhd*

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY registro\_2 IS

   PORT (

       entrada : IN std\_logic\_vector(1 downto 0);

       reset : IN std\_logic;

       clk : IN std\_logic;

       salida : OUT std\_logic\_vector(1 downto 0)

   );

END registro\_2;

ARCHITECTURE behavioral OF registro\_2 IS

SIGNAL reg : std\_logic\_vector(1 downto 0);

BEGIN

   PROCESS (clk, entrada) BEGIN

       IF rising\_edge(clk) THEN

           IF reset = '1' THEN

               reg <= "ZZ";

           ELSE

               reg <= entrada;

           END IF;

       END IF;

       salida <= reg;

   END PROCESS;

END behavioral;

*Figura 7.11 Código registro\_2.vhd*

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

ENTITY registro\_4 IS

   PORT (

       entrada : IN std\_logic\_vector(3 downto 0);

       reset : IN std\_logic;

       clk : IN std\_logic;

       salida : OUT std\_logic\_vector(3 downto 0)

   );

END registro\_4;

ARCHITECTURE behavioral OF registro\_4 IS

SIGNAL reg : std\_logic\_vector(3 downto 0);

BEGIN

   PROCESS (clk, entrada) BEGIN

       IF rising\_edge(clk) THEN

           IF reset = '1' THEN

               reg <= "ZZZZ";

           ELSE

               reg <= entrada;

           END IF;

       END IF;

       salida <= reg;

   END PROCESS;

END behavioral;

*Figura 7.11 Código registro\_4.vhd*

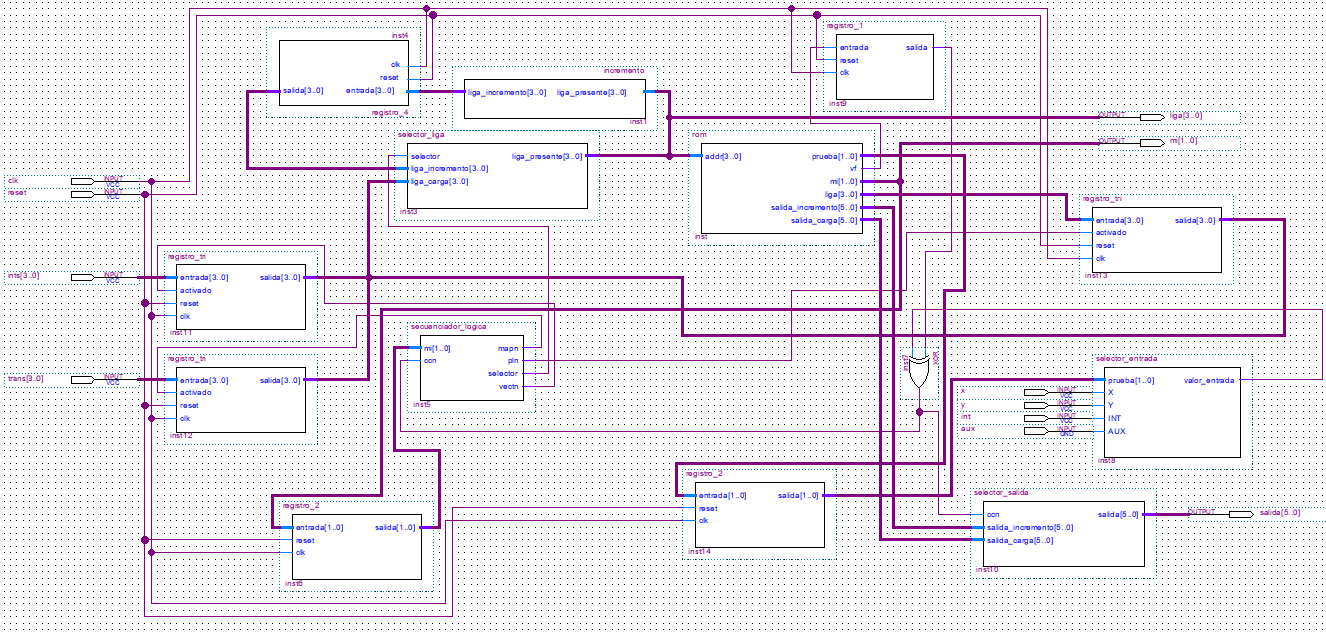
Una vez realizado cada uno de los componentes necesarios para llevar a cabo cada una de las partes para el secuenciador, obtuvimos cada uno de sus símbolos para poder unirlos en un esquema, obteniendo el siguiente resultado:

***rom***

***registro\_1***

***registro\_4***

***rom***



***registro\_tri***

***registro\_tri***

***selector\_liga***

***selector\_entrada***

***selector\_salida***

***secuenciador***

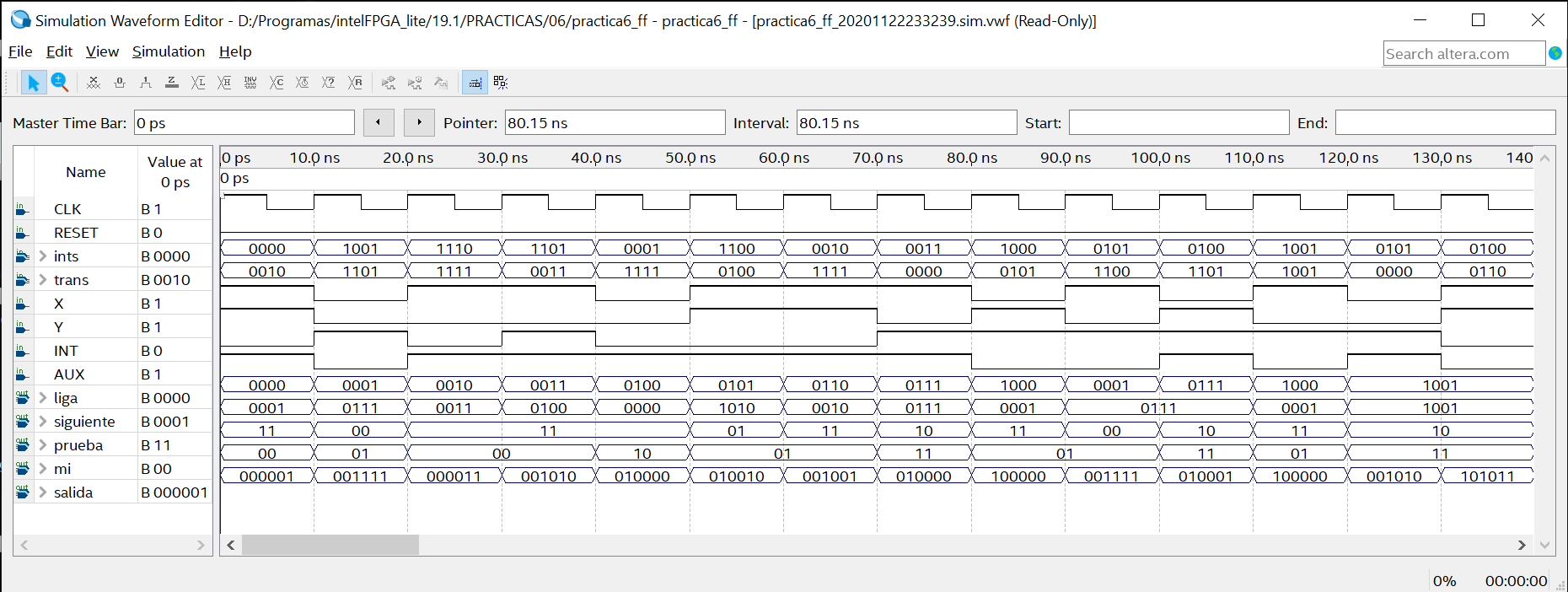
***registro\_2***

***registro\_2***

***registro\_tri***

*Figura 7.12 Esquema del secuenciador básico*

Como podemos observar se sigue prácticamente el diagrama mostrado en la introducción que pertenece al direccionamiento implícito. Ahora bien, una vez compilado el bloque entero y asignado cada una de las entradas, así como la señal de reloj y de Reset, procedemos a simularlo



*Figura 7.13 Simulación del Esquemático*

Como podemos ver en la simulación se observan las señales de reloj (*clk*), reset, entradas (*X, Y, INT, AUX*), ints y trans (que nos ayudaron a activar los registros), estado actual (*liga*), estado siguiente (*siguiente*), el valor de prueba (*prueba*), el valor de la microinstrucción utilizada por el secuenciador (*mi)* y finalmente las salidas (*salida*).

Siguiendo la carta ASM y la tabla de verdad junto con los valores de las entradas y de las microinstrucciones, podemos comprobar que efectivamente se lleva a cabo de manera correcta el funcionamiento secuenciador básico. Arrojando así todos los valores de las salidas correspondientes al estado que se está analizando. Por lo que la simulación es correcta ya que nos indica de buena manera el estado siguiente al que se debe de ir y la salida que se obtuvo con esta.

# **Fuentes de Consulta**

* Savage, J (2015) Diseño de microprocesadores. Facultad de Ingeniería. Universidad Nacional Autónoma de México. 482pp,
* Chávez, N (S.F) Construcción de máquinas de estados usando memorias. Consultado el 23 de noviembre del 2020 de: <http://profesores.fi-b.unam.mx/normaelva/Direccionamientos.pdf>