UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

LABORATORIO DE ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORAS

ING. ADRIAN ULISES MERCADO MARTÍNEZ

GRUPO 4

PRÁCTICA 6: Secuenciador básico

CUENCA PALACIOS ISRAEL

GARCÍA LAZCANO CARLOS DAVID

OBJETIVO.

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

INTRODUCCIÓN

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.
La figura 4.1 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: 1) del registro µPC, ó 2) de la entrada D.
1. El registro de micro-programa (µPC) contiene la dirección del estado presente más uno, es decir, la dirección que se encuentra a la salida del multiplexor es incrementada en una unidad y cargada en este registro en el siguiente ciclo de reloj.
2. En la entrada D se introduce una dirección de salto. Esta dirección puede venir de tres lugares diferentes: del campo de liga, del registro de transformación o del registro de interrupciones.

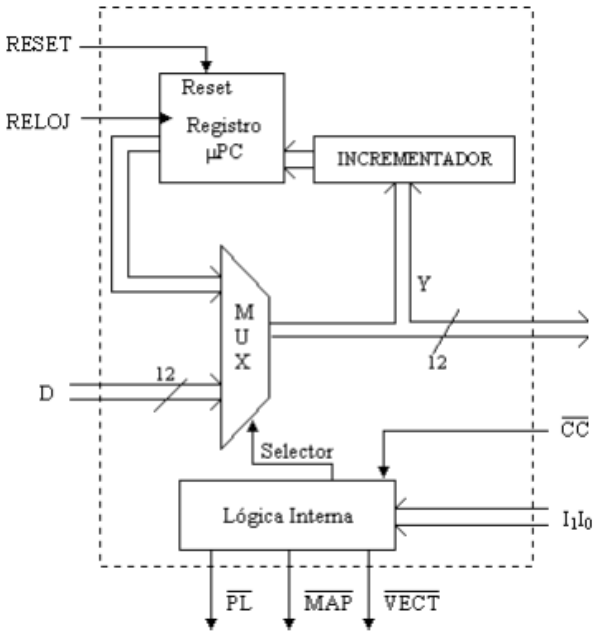


Figura 4.1. Diagrama de bloques del secuenciador básico.

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 µPC o la entrada D. Dicha salida direcciona una memoria que contiene el estado siguiente del algoritmo de la máquina de estados. 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. Esta característica se utilizará cuando se diseñe la unidad central de procesos (UCP), como se verá más adelante.
La figura 4.2 muestra las señales de entrada y salida del secuenciador.

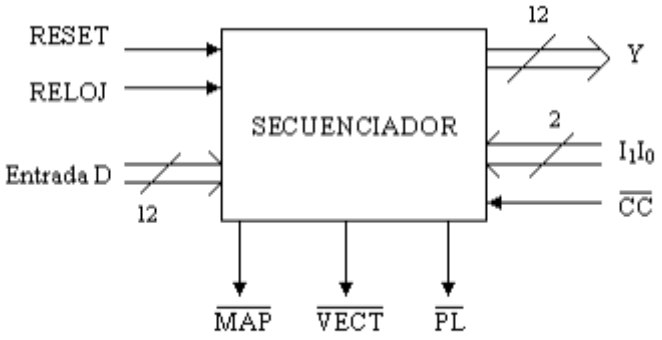


Figura 4.2. Secuenciador básico.

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

INSTRUCCIONES PARA EL SECUENCIADOR

CONTINÚA ©

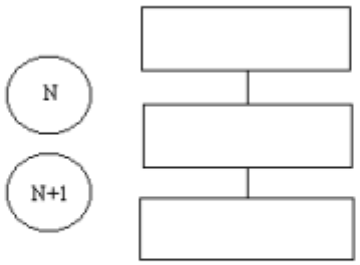


Figura 4.3. Representación en notación ASM de la instrucción continúa.

Figura 4.3. Representación en notación ASM de la instrucción continúa.
En la instrucción continúa la dirección del estado siguiente la proporciona el registro μPC.

SALTO CONDICIONAL (SCO)

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 μPC; si es igual a cero, la dirección del estado siguiente, contenida en el registro seleccionado por PL , ingresa a través de la entrada D.

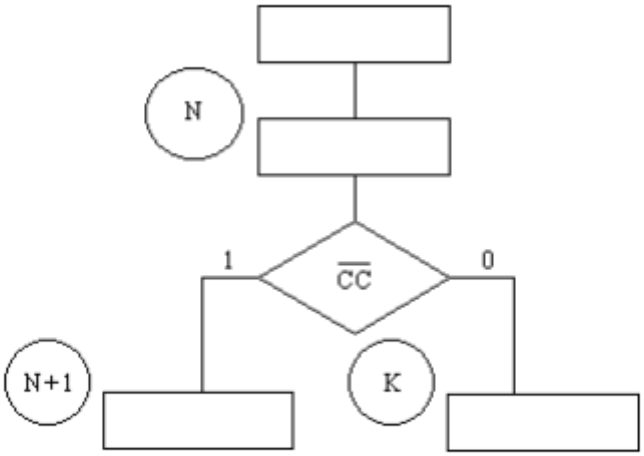


Figura 4.4. Representación en notación ASM de la instrucción SCO.

SALTO DE TRANSFORMACIÓN (ST)

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 .

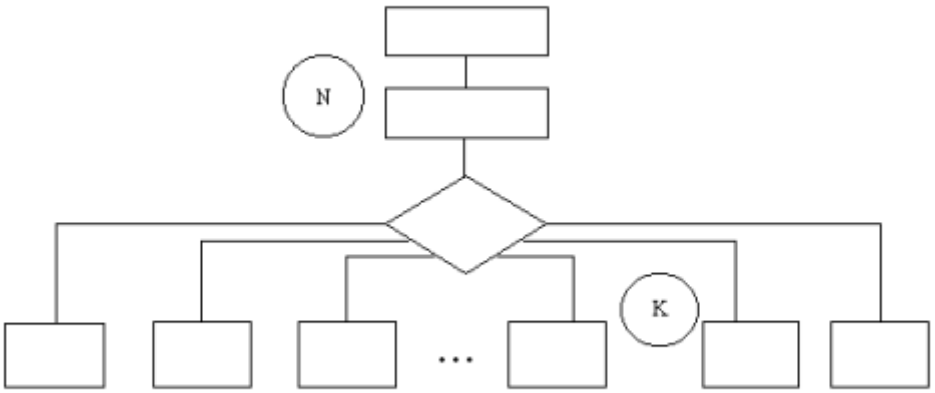


Figura 4.5. Representación en notación ASM de la instrucción ST.

SALTO CONDICIONAL USANDO LA DIRECCIÓN DE LAS

INTERRUPCIONES (SCI)

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

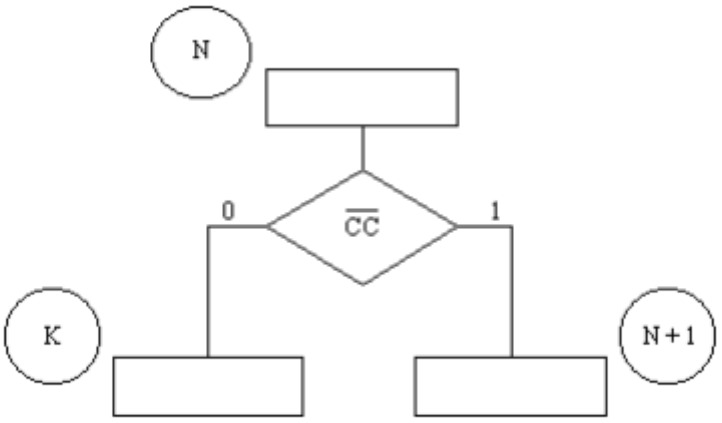


Figura 4.6. Representación en notación ASM de la Instrucción SCI.

La lógica interna del secuenciador se construye a partir de la siguiente tabla.

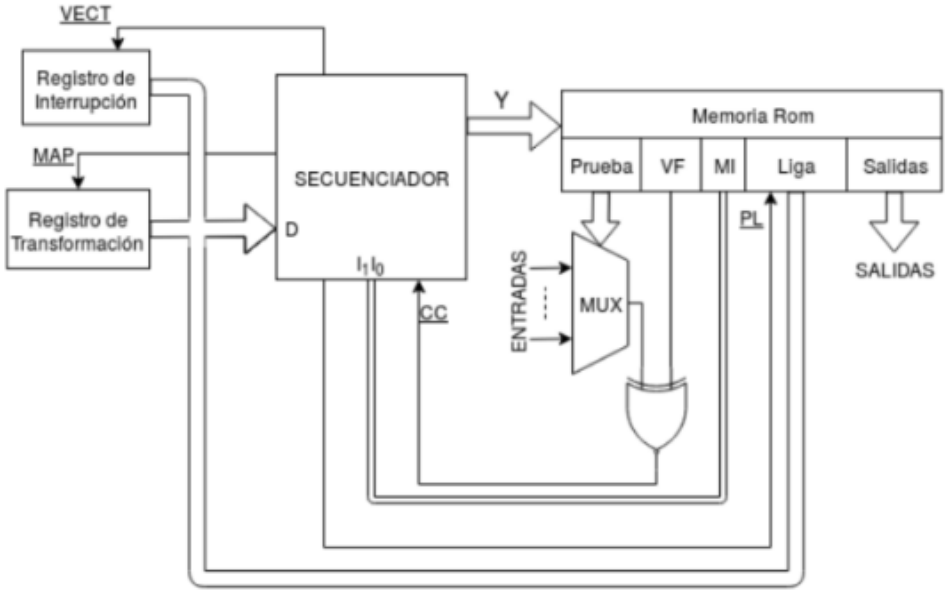
Entradas			**Salidas**				
I1	I0	CC	Selector	PL	MAP	VECT	Y
0	0	0	0	1	1	1	µPC
0	0	1	0	1	1	1	µPC
0	1	0	1	0	1	1	Entrada D
0	1	1	0	0	1	1	µPC
1	0	0	1	1	0	1	Entrada D
1	0	1	1	1	0	1	Entrada D
1	1	0	1	1	1	0	Entrada D
1	1	1	0	1	1	0	µPC

Tabla. 4.1. Entradas y salidas de la lógica interna del secuenciador.

DESARROLLO.

1. La figura 4.1, muestra el diagrama de bloques de un secuenciador básico. Como se puede observar en el diagrama, la dirección del estado siguiente dada por el bus Y puede venir de dos lugares posibles: 1 del registro µP C, 2 de la entrada D.
El registro de microprograma contiene la dirección del estado presente más uno, es decir, la dirección que se encuentra a la salida del multiplexor es incrementada en una unidad y cargada en este registro en el siguiente ciclo de reloj.
En la entrada D se introduce una dirección de salto. Esta dirección puede venir de tres lugares diferentes: del campo de liga, del registro de transformación o del registro de interrupción.

2. La figura siguiente, muestra el diagrama de bloques de un secuenciador básico conectado a una memoria.



Construya el secuenciador descrito en el punto 1 y conéctelo a una memoria tal como se muestra en el punto 2, utilizando VHDL y componentes estándares del software de desarrollo. Diseñe la lógica interna para que su secuenciador pueda ejecutar las instrucciones descritas anteriormente. Obtenga el contenido de la memoria de la carta ASM e implemente en su secuenciador. Utilice el simulador para probar su implementación.
ASM:

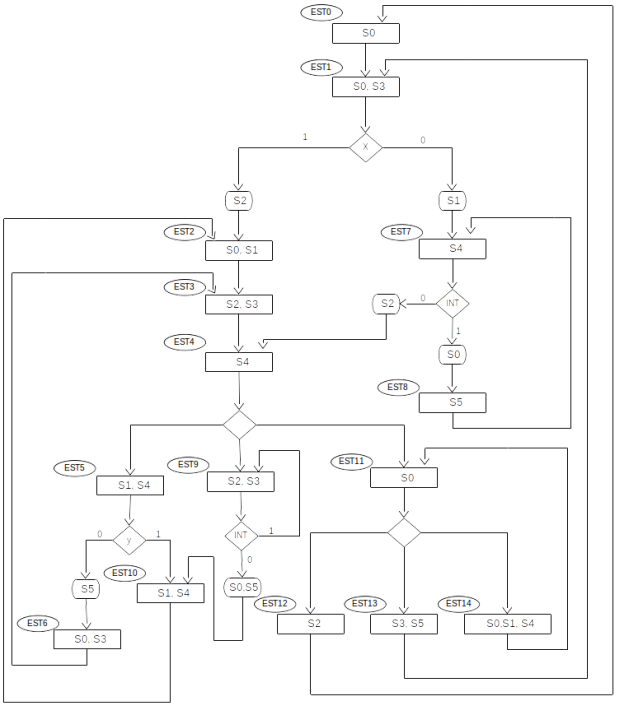


Tabla de verdad:

direccion				contenido de memoria																			
estado presente				prueba		vf		mi		liga				salidas verdaderas						salidas falsas					
0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1
0	0	0	1	0	0	0	0	1	0	1	1	1	0	0	1	1	0	1	0	0	1	0	1	1
0	0	1	0	1	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1
0	0	1	1	1	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0
0	1	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	1	1	0	1	1	0	1	0	0	1	0	0	1	0	1	1	0	0	1	0
0	1	1	0	1	0	0	0	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1
0	1	1	1	1	1	0	1	1	0	1	1	1	0	1	0	0	0	1	0	1	0	1	0	0
1	0	0	0	1	0	0	0	1	0	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1	0	0	0	0	0	1	1	0	0	1	0	1	1	0	1
1	0	1	0	1	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
1	0	1	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0	1	0	0	0	0	0	1
1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
1	1	0	1	1	0	0	0	1	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0
1	1	1	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1	1	0	1	0	0	1	1
1	1	1	1																					

Mif:

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	1000000010000010000001	0000101110011010010111	1000000110000110000011	1000001000011000001100	1001001000100000010000	0110110100100101100010	1000100110010010010001	1101101110100010101000	.....
8	1000101111000001000000	1111111000001100101101	1000100100100100100010	1001010110000010000001	1000100000001000000100	1000100001010001010000	1000100000100110100111		.....

Micro pc:

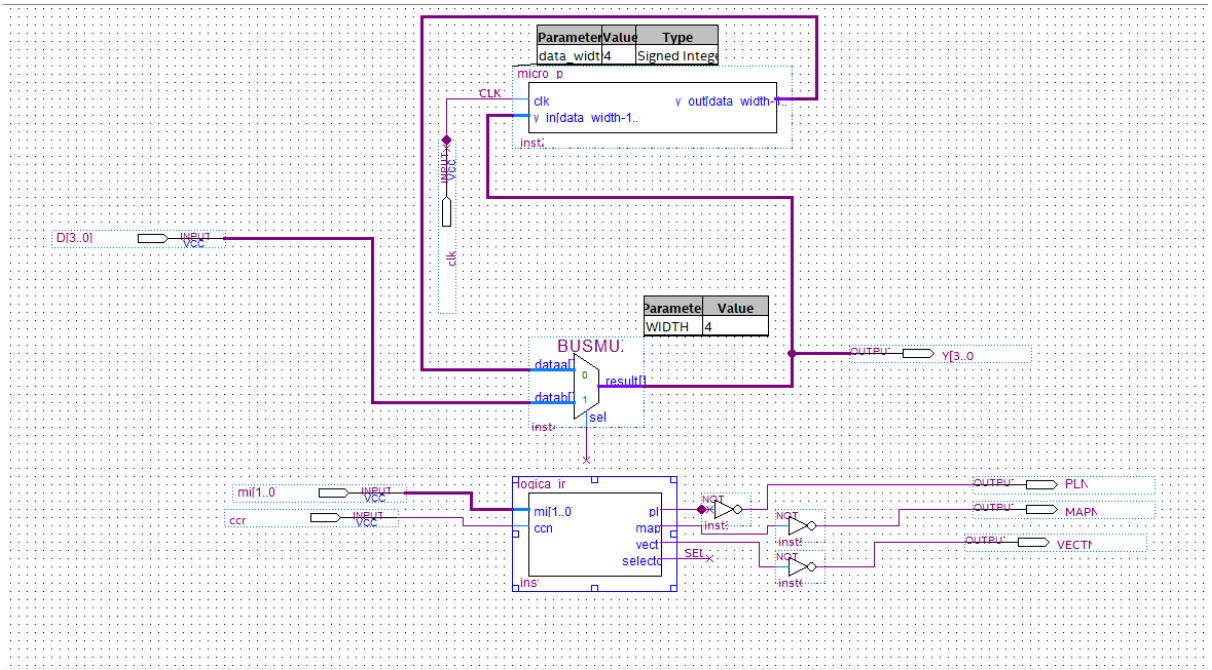
```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity micro_pc is
	generic(
		data_width: natural :=3);
	port(
	clk: in std_logic;
	y_in : in std_logic_vector(data_width-1 downto 0);
	y_out :out std_logic_vector (data_width-1 downto 0));
end entity micro_pc;

architecture behavorial of micro_pc is
	signal count: integer range 0 to 2**data_width-1 :=0;
begin
	process(clk)
	begin
		if rising_edge(clk) then
			count <= to_integer(unsigned(y_in))+1;
		end if;
	end process;
	y_out <= std_logic_vector(to_unsigned(count, y_out'length));
end architecture;
```

lógica interna:

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

```
entity logica_int is
	port(
		mi : in std_logic_vector(1 downto 0);
		ccn: in std_logic;
		pl : out std_logic;
		mapl: out std_logic;
		vect: out std_logic;
		selector: out std_logic);
end logica_int;

architecture behavorial of logica_int is
begin
	process(mi, ccn)
	begin
		case mi is
			when "00"=>
				selector <= '0';
				pl <= '1';
				mapl <= '1';
				vect <= '1';
				when "01"=>
					if ccn= '0' then
						selector <= '0';
						pl <= '1';
						mapl <= '1';
						vect <= '1';
					else
						selector <= '1';
						pl <= '0';
						mapl <= '1';
						vect <= '1';
					end if;
				when "10"=>
					selector <= '1';
					pl <= '1';
					mapl <= '0';
					vect <= '1';
				when "11"=>
					if ccn= '0' then
						selector <= '0';
						pl <= '1';
						mapl <= '1';
						vect <= '1';
					else
						selector <= '1';
						pl <= '1';
						mapl <= '1';
						vect <= '0';
					end if;
		end case;
	end process;
end architecture;
```

Secuenciador:

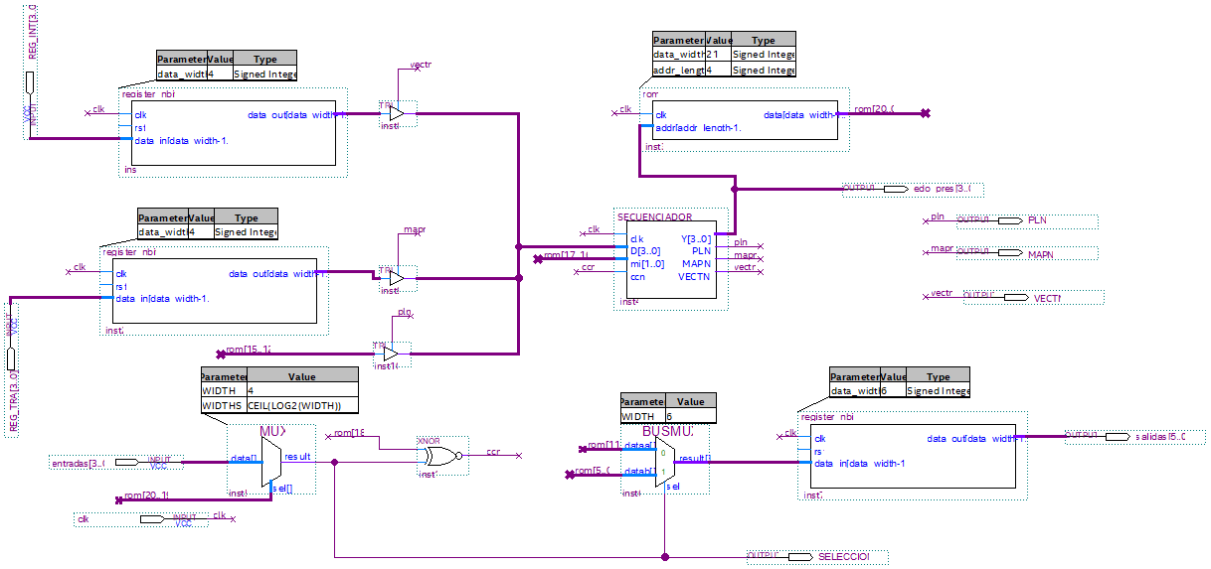


registro nbist:

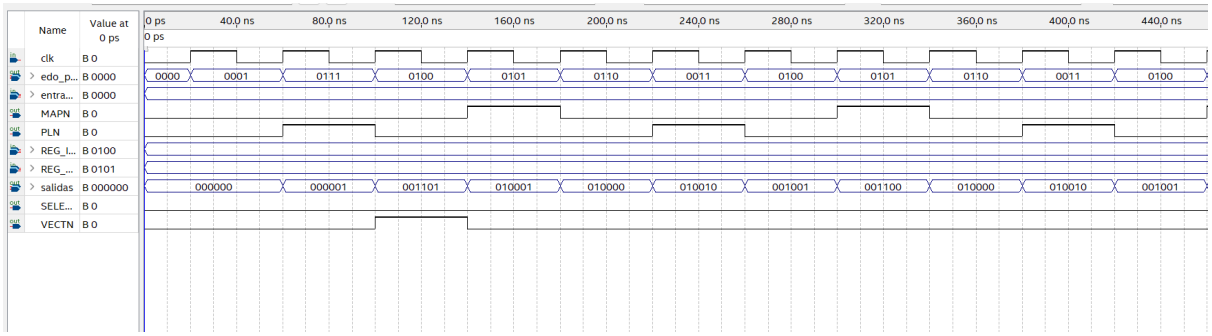
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity register_nbits is
	generic(
		data_width: integer:=3
	);
	port(
		clk:in std_logic;
		rst: in std_logic;
		data_in: in std_logic_vector(data_width-1 downto 0);
		data_out: out std_logic_vector(data_width-1 downto 0)
	);
end entity;

architecture behavioral of register_nbits is
--signal data: std_logic_vector(data_width-1 downto 0);
begin
	process(clk)
	begin
		if rst = '1' then
			data_out <= std_logic_vector(to_unsigned(0, data_out'length));
		elsif rising_edge (clk) then
			data_out <= data_in;
		end if;
	end process;
end behavioral;
```

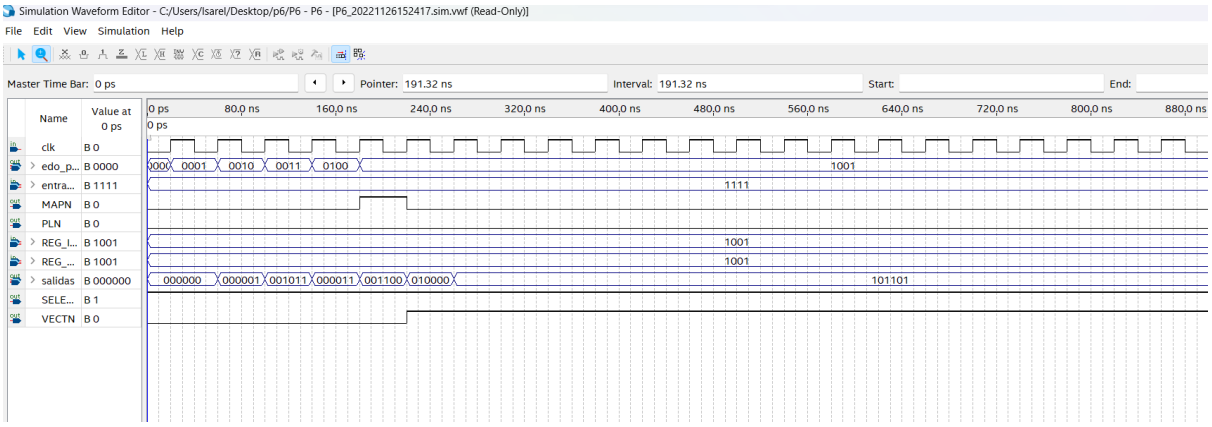
Rom:

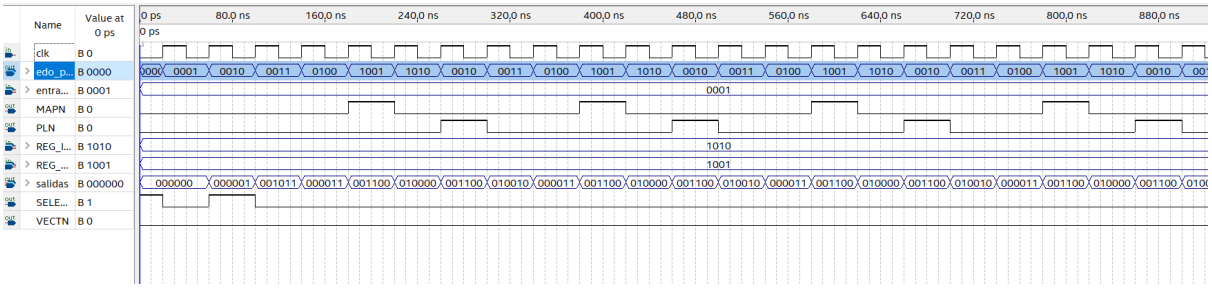
```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rom is
	generic(
		data_width: integer :=8;
		addr_length: integer :=3);
	port(
	clk: in std_logic;
	addr: in std_logic_vector(addr_length-1 downto 0);
	data: out std_logic_vector(data_width-1 downto 0));
end entity rom;

architecture behavorial of rom is
	type memory is array(2**addr_length-1 downto 0) of std_logic_vector(data_width-1 downto 0);
	signal rom_data: memory;
	attribute ram_init_file: string;
	attribute ram_init_file of rom_data : signal is "mifbuenop6.mif";
	signal datum: std_logic_vector(data_width-1 downto 0);
begin
	process(addr)
	begin
		datum <= rom_data(to_integer(unsigned(addr)));
	end process;

	process (datum, clk)
	begin
		if rising_edge(clk) then
			data <= datum;
		end if;
	end process;
end architecture;
```


SECUENCIADOR
clk
D[3..0]
mi[1..0]
ccn
Y[3..0]
PLN
MAPN
VECTN
MUX
BUSMUX
WIDTH 4
WIDTHS CEIL(LOG2(WIDTH))
WIDTH 6
PLN
MAPN
VECTI
SELECCIO

simulación;


Name
Value at 0 ps
clk
edo_p...
entra...
MAPN
PLN
REG_I...
REG_...
salidas
SELE...
VECTN
0000 0001 0111 0100 0101 0110 0011 0100 0101 0110 0011 0100
000000 000001 001101 010001 010000 010010 001001 001100 010000 010010 001001


Simulation Waveform Editor - C:/Users/Isarel/Desktop/p6/P6 - P6 - [P6_20221126152417.sim.vwf (Read-Only)]
File Edit View Simulation Help
Master Time Bar: 0 ps
Pointer: 191.32 ns
Interval: 191.32 ns
Start:
End:
Name
Value at 0 ps
clk
edo_p...
entra...
MAPN
PLN
REG_I...
REG_...
salidas
SELE...
VECTN
0000 0001 0010 0011 0100 1001
1111
1001
1001
000000 000001 001011 000011 001100 010000 101101



CONCLUSIONES.

Cuenca Palacios Israel:
Para la realización de está práctica fue un poco tedioso, debido a que se tomaba en cuenta código previamente implementado en prácticas anteriores, además de un error en el diseño que hacía que al momento de simular se saltara varios estados, sólo se hacía medio recorrido, pero por suerte solo era una compuerta la que causaba este comportamiento y se pudo resolver, de esta manera, se pueden ver cada una de las funcionalidades de los saltos condicionales, de transformación y de interrupción,por lo tanto el objetivo de la práctica fue cumplido satisfactoriamente, por el hecho de comprender cómo funciona un secuenciador básico y lo que lo conforman..

García Lazcano Carlos David:
En esta práctica fue necesario tener ideas básicas de lo que es un secuenciador básico, usando esta vez miPc, vectn, pl y mapn para poder usar correctamente cada uno de los elementos para no tener errores, de hacerlas, podemos quemar nuestro secuenciador, aunque sea solo simulado. Para este trabajo se necesitaron cada uno de los códigos previamente usados como base para trabajar, de modo que si habían carencias del concepto, en la implementación quedaron cubiertas. Sin embargo nos aquejaron varios errores que hacia que nuestra simulación no funcionara como se esperaba, es decir, hacia cosas que no debía, se le atribuía errores al mif puesto que se siguió el diseño proporcionado, por suerte el diseño tenía un error en una compuerta xor que debía de ser xnor y con ello se soluciono el error .Por ello se pudo observar el comportamiento del secuenciador, por lo que los objetivos fueron cumplidos.

Equipo:
La implementación fue sencilla, así como la creación del MIF, puesto que contamos con la ayuda del profesor para elaborarla, por lo que los conceptos y la abstracción fueron bien comprendidos, de una manera adecuada, sin dejar de lado unos pequeños errores que obstaculizaron el término de la práctica en un tiempo prolongado extra a lo esperado. por ello quedamos satisfechos con los resultados de dicha práctica de secuenciador básico.

REFERENCIAS.

(N.d.-a). Unam.Mx. Retrieved November 21, 2022, from

https://biorobotics.fi-p.unam.mx/wp-content/uploads/Courses/laboratorio_

de_organizacion_y_arquitectura_de_computadoras/2020-2/practicas/practica6_oac.pdf

(N.d.-b). Unam.Mx. Retrieved November 21, 2022, from https://biorobotics.fi-p.unam.mx/wp-content/uploads/Courses/arquitectura_de_computadoras/2021-1/lecciones/Leccion_4_arquitectura_de_computadoras_2021_1.pdf