Práctica Final Laboratorio de Circuitos Lógicos

Program Counter

Rodrigo Plauchú Rodríguez

CU: 182671

**Objetivos**

Realizar la implementación de un Program Counter utilizando VHDL, reforzar la diferencia entre circuitos secuenciales y combinacionales, así como implementar el program counter de la computadora de un solo ciclo, así como los componentes asociados al program counter.

**Introducción**

Un contador de programa (Program Counter en inglés) es un registro interno de la computadora en cual se almacena la dirección de la última instrucción leída. De esta manera la computadora puede saber cuál es la siguiente instrucción que debe ejecutar. El Program Counter va incrementándose en una unidad, salvo que la instrucción ejecutándose en ese instante cambie. La instrucción puede ser de un byte, 2, o más, según el objetivo que se busque y el tamaño de selección.

Para la implementación del PC podemos reutilizar muchos componentes de anteriores prácticas, como por ejemplo la práctica 2, la práctica 4, el MUX 4-1, MUX 2-1, deco. 2-4, div. De frecuencia, el BCD a 7 segmentos, shifter, elementos de la práctica 5, entre otros. Todos estos fueron incluidos en el archivo de la práctica final. Adicionado a estos, se solicitó la implementación de otros componentes.

**Desarrollo**

Adicionado a estos, se solicitó la implementación de otros componentes.

El Branch Control, básicamente recibe las señales:

* PL = 1 - Indica si se debe realizar una operación Jump o Branch.
* JB = 1 - Indica que la operación es Jump.
* JB = 0 - Indica que la operación es Branch.
* BC = 1 - Indica que la operación a realizar es Conditional Branch on Negative.
* BC = 0 - Indica que la operación a realizar es Conditional Branch on Zero.
* N - Indica si la operación anterior resultó en un número negativo.
* Z - Indica si la operación anterior resultó en cero.

En donde utilizando estas señales de entrada debemos generar una de salida que le indique al PC qué tipo de operación realizar. Se sigue además un formato indicado para las operaciones branch y jump.

El código en Vhdl y sus operaciones booleanas son las siguientes:

library ieee ;

use ieee.std\_logic\_1164.all ;

use ieee.numeric\_std.all;

library work;

ENTITY branch\_control is

port (

PL, JB, BC, N, Z : in STD\_LOGIC;

oper : out STD\_LOGIC\_VECTOR(1 DOWNTO 0)

);

END branch\_control;

ARCHITECTURE bdf\_type of branch\_control is

signal res: STD\_LOGIC\_VECTOR (1 DOWNTO 0) := "00";

begin

res(0) <= (PL AND BC AND N AND (JB XOR (PL AND BC AND N))) OR (PL AND Z AND ((JB AND BC) XOR (PL AND Z)));

res(1) <= PL AND JB;

oper <= res;

end bdf\_type;

Para el PC se define para realizar 3 operaciones:

* PC ←PC + 1: Para las operaciones normales
* PC ←PC + se AD: Para las operaciones branch.
* PC ←R[SA]: Para la operación jump.

En donde justamente depende de la salida del Branch Control que funciona como selector para indicar qué tipo de operación se debe realizar.

El código en Vhdl y sus operaciones booleanas son las siguientes:

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_signed.all;

use ieee.std\_logic\_arith.all;

library work;

ENTITY program\_counter is

port (

CLK : IN STD\_LOGIC;

OP : IN STD\_LOGIC\_VECTOR(1 DOWNTO 0);

AD : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);

A\_DATA : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);

OUT\_PC : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0)

);

END program\_counter;

ARCHITECTURE bdf\_type of program\_counter is

SIGNAL PC : STD\_LOGIC\_VECTOR(15 DOWNTO 0) := "0000000000000000";

begin

PROCESS (CLK)

begin

if (CLK'event and (CLK ='1')) then

case OP is

when "00" => PC <= PC + 1;

when "01" => PC <= PC + AD;

when "10" => PC <= A\_DATA;

when others => PC <= "XXXXXXXXXXXXXXXX";

end case;

end if;

OUT\_PC <= PC;

end process;

end bdf\_type;

Para el Extend es un componente que como indica su nombre extiende un vector de 6 a 16 bits. Precisamente para una operación: PC ←PC + se AD.

AD es un número signado mediante complemento a 2.

El código en Vhdl y sus operaciones booleanas son las siguientes:

library ieee ;

use ieee . std\_logic\_1164 . all ;

use ieee . numeric\_std . all ;

entity extend is

port

(

par8\_6: in std\_logic\_vector (2 downto 0);

par2\_0: in std\_logic\_vector (2 downto 0);

res : out std\_logic\_vector (15 downto 0)

);

end extend;

architecture extend of extend is

begin

with par8\_6(2) select

res <=

"00000000000" & par8\_6(1 downto 0) & par8\_6(2 downto 0) when '0',

"11111111111" & par8\_6(1 downto 0) & par8\_6(2 downto 0) when others;

end extend ;

Ahora justamente se adaptan ciertos aspectos técnicos de prácticas pasadas, particularmente para probar el funcionamiento del PC, utilizaremos la salida de este a la entrada del BCD a 7 segmentos en hexadecimal. Se solicitó además utilizar un botón de FPGA para el reloj de entrada y para las demás entradas los dip switches.

Después, falta por implementar el Zero Fill, que simplemente recibe un número de 3 bits en entrada y entrega ese mismo número en su representación hexadecimal, 16 bits, esto se logra concatenando 0’s a la izquierda del número original.

El código en Vhdl y sus operaciones booleanas son las siguientes:

library ieee ;

use ieee.std\_logic\_1164.all ;

use ieee.std\_logic\_unsigned.all ;

use ieee.std\_logic\_arith.all ;

use ieee.numeric\_std ;

entity zero\_fill is

port

(

z\_in : in std\_logic\_vector (2 downto 0);

z\_out : out std\_logic\_vector (15 downto 0)

);

end zero\_fill;

architecture function\_table of zero\_fill is

begin

z\_out <= "0000000000000" & z\_in;

end function\_table ;

Posteriormente, se implementa el Instruction Decoder con ayuda de un diagrama en donde se programan compuertas and y nor con un vector que da la instrucción de 16 bits y una palabra control de 20 bits y lo asigna a 3 compuertas lógicas, así como a la palabra control.

El código en Vhdl y sus operaciones booleanas son las siguientes:

library ieee ;

use ieee.std\_logic\_1164.all ;

use ieee.std\_logic\_unsigned.all ;

use ieee.std\_logic\_arith.all ;

use ieee.numeric\_std ;

entity instruction\_decoder is

port

(

instruction : in std\_logic\_vector (15 downto 0);

control\_word : out std\_logic\_vector (19 downto 0)

);

end instruction\_decoder;

architecture instruction\_decoder of instruction\_decoder is

signal gate\_1 : std\_logic;

signal gate\_2 : std\_logic;

signal gate\_3 : std\_logic;

begin

gate\_1 <= instruction(9) and not (instruction(14) and instruction(15));

gate\_2 <= instruction(14) and not (instruction(15));

gate\_3 <= instruction(14) and instruction(15);

control\_word <= instruction(8 downto 6)

& instruction(5 downto 3)

& instruction(2 downto 0)

& instruction(15)

& instruction(12 downto 10)

& gate\_1

& instruction(13)

& not (instruction(14))

& gate\_2

& gate\_3

& instruction(13)

& instruction(9);

end instruction\_decoder ;

Finalmente, se implementa el Instruction Specifications, en este componente es parecido al anterior pero ahora son 3 instrucciones y sus especificaciones ahora se muestran en una tabla con 19 instrucciones distintas, es importante mencionar que en todas estas instrucciones también se ejecuta PC ←PC + 1 para preparar el siguiente ciclo.

El código en Vhdl y sus operaciones booleanas son las siguientes:

library ieee;

use ieee.std\_logic\_1164.all ;

entity instruction\_memory is

port

(

CLK, RESET: in std\_logic;

IE : in std\_logic\_vector(15 downto 0);

I : out std\_logic\_Vector(15 downto 0)

);

end instruction\_memory;

architecture flujo1 of instruction\_memory is

signal R0 : std\_logic\_vector(2 downto 0) := "000"; --REGISTRO '0'

signal R1 : std\_logic\_vector(2 downto 0) := "001"; --REGISTRO '1'

signal R2 : std\_logic\_vector(2 downto 0) := "010"; --REGISTRO '2'

signal R3 : std\_logic\_vector(2 downto 0) := "011"; --REGISTRO '3'

signal R4 : std\_logic\_vector(2 downto 0) := "100"; --REGISTRO '4'

signal R5 : std\_logic\_vector(2 downto 0) := "101"; --REGISTRO '5'

signal R6 : std\_logic\_vector(2 downto 0) := "110"; --REGISTRO '6'

signal R7 : std\_logic\_vector(2 downto 0) := "111"; --REGISTRO '7'

signal MOVA : std\_logic\_vector(6 downto 0) := "0000000"; --RD, RA

signal INC : std\_logic\_vector(6 downto 0) := "0000001"; --RD, RA

signal ADD : std\_logic\_vector(6 downto 0) := "0000010"; --RD, RA, RB

signal SUB : std\_logic\_vector(6 downto 0) := "0000101"; --RD, RA, RB

signal DEC : std\_logic\_vector(6 downto 0) := "0000110"; --RD, RA

signal ANDD : std\_logic\_vector(6 downto 0) := "0001000"; --RD, RA, RB

signal OOR : std\_logic\_vector(6 downto 0) := "0001001"; --RD, RA, RB

signal ORX : std\_logic\_vector(6 downto 0) := "0001010"; --RD, RA, RB

signal NNO : std\_logic\_vector(6 downto 0) := "0001011"; --RD, RA

signal MOVB : std\_logic\_vector(6 downto 0) := "0001100"; --RD, RB

signal SHR : std\_logic\_vector(6 downto 0) := "0001101"; --RD, RB

signal SHL : std\_logic\_vector(6 downto 0) := "0001110"; --RD, RB

signal LDI : std\_logic\_vector(6 downto 0) := "1001100"; --RD, OP

signal ADI : std\_logic\_vector(6 downto 0) := "1000010"; --RD, RA, OP

signal LD : std\_logic\_vector(6 downto 0) := "0010000"; --RD, RA

signal ST : std\_logic\_vector(6 downto 0) := "0100000"; --RD, RB

signal BRZ : std\_logic\_vector(6 downto 0) := "1100000"; --RD, AD

signal BRN : std\_logic\_vector(6 downto 0) := "1100001"; --RD, AD

signal JMP : std\_logic\_vector(6 downto 0) := "1110000"; --RA

BEGIN

R0 <= "000"; --REGISTRO '0'

R1 <= "001"; --REGISTRO '1'

R2 <= "010"; --REGISTRO '2'

R3 <= "011"; --REGISTRO '3'

R4 <= "100"; --REGISTRO '4'

R5 <= "101"; --REGISTRO '5'

R6 <= "110"; --REGISTRO '6'

R7 <= "111"; --REGISTRO '7'

MOVA <= "0000000"; --RD, RA

INC <= "0000001"; --RD, RA

ADD <= "0000010"; --RD, RA, RB

SUB <= "0000101"; --RD, RA, RB

DEC <= "0000110"; --RD, RA

ANDD <= "0001000"; --RD, RA, RB

OOR <= "0001001"; --RD, RA, RB

ORX <= "0001010"; --RD, RA, RB

NNO <= "0001011"; --RD, RA

MOVB <= "0001100"; --RD, RB

SHR <= "0001101"; --RD, RB

SHL <= "0001110"; --RD, RB

LDI <= "1001100"; --RD, OP

ADI <= "1000010"; --RD, RA, OP

LD <= "0010000"; --RD, RA

ST <= "0100000"; --RD, RB

BRZ <= "1100000"; --RD, AD

BRN <= "1100001"; --RD, AD

JMP <= "1110000"; --RA

I <= "0000000" & O"0" & O"0" & O"0" WHEN RESET = '1' ELSE

-- Éste programa carga dos valores de data memory a los registros R0 y R1 para operar

-- con estos valores y guardar el resultado de cada operación en los registros restantes

------------------------ STORE-DISPLAY TEST --------------------------------------

LDI & O"0" & O"0" & O"0" when IE = X"0000" else -- Se asigna una constante ? al registro 0

LD & O"0" & O"0" & O"0" when IE = X"0001" else -- Se carga de data memory el valor en el registro ?

LDI & O"1" & O"0" & O"1" when IE = X"0002" else -- Se asigna una constante ! al registro 1

LD & O"1" & O"1" & O"0" when IE = X"0003" else -- Se carga de data memory el valor en el registro !

ADD & O"2" & O"0" & O"1" when IE = X"0004" else

ORX & O"3" & O"0" & O"1" when IE = X"0005" else

OOR & O"4" & O"2" & O"3" when IE = X"0006" else

SHL & O"5" & O"0" & O"3" when IE = X"0007" else

NNO & O"6" & O"3" & O"0" when IE = X"0008" else

ORX & O"7" & O"3" & O"1" when IE = X"0009" else

------------------------ STORE-DISPLAY TEST --------------------------------------

"0101010" & "101" & "010" & "101" ;

end flujo1;

Para finalizar, lo único que falta es realizar el componente final, en donde se declaran todas las entidades que se van a usar, no todos los archivos .vhd que están en el documento se utilizaron, sin embargo, era mejor tenerlos todos juntos, en este caso se especifica los 2 clocks, el reset, 1 dip y 5 salidas. Y con ayuda de señales se empieza a armar la arquitectura para poder conectarlos en el esquemático. Una vez terminado y generando los bloques se conectan todos los componentes y queda armado nuestro PC, que simplemente utiliza la lógica de realizar las instrucciones que se le piden, operaciones, micro-operaciones y otros procesos por medio de una palabra de control como un selector.

El código en Vhdl y sus operaciones booleanas son las siguientes:

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

LIBRARY work;

ENTITY Final IS

PORT

(

CLK\_BTN : IN STD\_LOGIC;

CLK : IN STD\_LOGIC;

RESET : IN STD\_LOGIC;

DIP1 : IN STD\_LOGIC\_VECTOR(2 DOWNTO 0);

OUTPUT1 : OUT STD\_LOGIC;

OUTPUT2 : OUT STD\_LOGIC;

OUTPUT3 : OUT STD\_LOGIC;

OUTPUT4 : OUT STD\_LOGIC;

sal : OUT STD\_LOGIC\_VECTOR(6 DOWNTO 0)

);

END Final;

ARCHITECTURE bdf\_type OF Final IS

COMPONENT program\_counter

PORT(CLK : IN STD\_LOGIC;

A\_DATA : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);

AD : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);

OP : IN STD\_LOGIC\_VECTOR(1 DOWNTO 0);

OUT\_PC : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0)

);

END COMPONENT;

COMPONENT extend

PORT(par2\_0 : IN STD\_LOGIC\_VECTOR(2 DOWNTO 0);

par8\_6 : IN STD\_LOGIC\_VECTOR(2 DOWNTO 0);

res : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0)

);

END COMPONENT;

COMPONENT instruction\_memory

PORT(CLK : IN STD\_LOGIC;

RESET : IN STD\_LOGIC;

IE : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);

I : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0)

);

END COMPONENT;

COMPONENT instruction\_decoder

PORT(instruction : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);

control\_word : OUT STD\_LOGIC\_VECTOR(19 DOWNTO 0)

);

END COMPONENT;

COMPONENT unidad\_funcional

PORT(A : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);

B : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);

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

C : OUT STD\_LOGIC;

N : OUT STD\_LOGIC;

Z : OUT STD\_LOGIC;

F : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0)

);

END COMPONENT;

COMPONENT data\_memory

PORT(CLK : IN STD\_LOGIC;

RESET : IN STD\_LOGIC;

MW : IN STD\_LOGIC;

Address : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);

Data\_in : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);

Data\_out : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0)

);

END COMPONENT;

COMPONENT multiplexer\_2\_to\_1

PORT(S : IN STD\_LOGIC;

I0 : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);

I1 : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);

ZERO : OUT STD\_LOGIC;

Y : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0)

);

END COMPONENT;

COMPONENT hexadisplay

PORT(input\_1 : IN STD\_LOGIC;

dip\_1 : IN STD\_LOGIC\_VECTOR(0 TO 3);

dip\_2 : IN STD\_LOGIC\_VECTOR(0 TO 3);

dip\_3 : IN STD\_LOGIC\_VECTOR(0 TO 3);

dip\_4 : IN STD\_LOGIC\_VECTOR(0 TO 3);

output\_1 : OUT STD\_LOGIC;

output\_2 : OUT STD\_LOGIC;

output\_3 : OUT STD\_LOGIC;

output\_4 : OUT STD\_LOGIC;

sal : OUT STD\_LOGIC\_VECTOR(6 DOWNTO 0)

);

END COMPONENT;

COMPONENT register\_file

PORT(CLK : IN STD\_LOGIC;

RESET : IN STD\_LOGIC;

RW : IN STD\_LOGIC;

AA : IN STD\_LOGIC\_VECTOR(2 DOWNTO 0);

BA : IN STD\_LOGIC\_VECTOR(2 DOWNTO 0);

D\_data\_in : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);

DA : IN STD\_LOGIC\_VECTOR(2 DOWNTO 0);

REG\_ADD : IN STD\_LOGIC\_VECTOR(2 DOWNTO 0);

A\_data\_out : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0);

B\_data\_out : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0);

HEX\_OUT : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0)

);

END COMPONENT;

COMPONENT zero\_fill

PORT(z\_in : IN STD\_LOGIC\_VECTOR(2 DOWNTO 0);

z\_out : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0)

);

END COMPONENT;

COMPONENT branch\_control

PORT(PL : IN STD\_LOGIC;

JB : IN STD\_LOGIC;

BC : IN STD\_LOGIC;

N : IN STD\_LOGIC;

Z : IN STD\_LOGIC;

oper : OUT STD\_LOGIC\_VECTOR(1 DOWNTO 0)

);

END COMPONENT;

SIGNAL control\_word : STD\_LOGIC\_VECTOR(19 DOWNTO 0);

SIGNAL HEX\_OUT : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SIGNAL I : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_15 : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_1 : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_2 : STD\_LOGIC\_VECTOR(1 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_3 : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_16 : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_8 : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_9 : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_10 : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_11 : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_12 : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_13 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_14 : STD\_LOGIC;

BEGIN

b2v\_inst : program\_counter

PORT MAP(CLK => CLK\_BTN,

A\_DATA => SYNTHESIZED\_WIRE\_15,

AD => SYNTHESIZED\_WIRE\_1,

OP => SYNTHESIZED\_WIRE\_2,

OUT\_PC => SYNTHESIZED\_WIRE\_3);

b2v\_inst1 : extend

PORT MAP(par2\_0 => I(2 DOWNTO 0),

par8\_6 => I(8 DOWNTO 6),

res => SYNTHESIZED\_WIRE\_1);

b2v\_inst10 : instruction\_memory

PORT MAP(CLK => CLK\_BTN,

RESET => RESET,

IE => SYNTHESIZED\_WIRE\_3,

I => I);

b2v\_inst12 : instruction\_decoder

PORT MAP(instruction => I,

control\_word => control\_word);

b2v\_inst14 : unidad\_funcional

PORT MAP(A => SYNTHESIZED\_WIRE\_15,

B => SYNTHESIZED\_WIRE\_16,

fs => control\_word(9 DOWNTO 6),

N => SYNTHESIZED\_WIRE\_13,

Z => SYNTHESIZED\_WIRE\_14,

F => SYNTHESIZED\_WIRE\_10);

b2v\_inst15 : data\_memory

PORT MAP(CLK => CLK\_BTN,

RESET => RESET,

MW => control\_word(3),

Address => SYNTHESIZED\_WIRE\_15,

Data\_in => SYNTHESIZED\_WIRE\_16,

Data\_out => SYNTHESIZED\_WIRE\_11);

b2v\_inst16 : multiplexer\_2\_to\_1

PORT MAP(S => control\_word(10),

I0 => SYNTHESIZED\_WIRE\_8,

I1 => SYNTHESIZED\_WIRE\_9,

Y => SYNTHESIZED\_WIRE\_16);

b2v\_inst17 : multiplexer\_2\_to\_1

PORT MAP(S => control\_word(5),

I0 => SYNTHESIZED\_WIRE\_10,

I1 => SYNTHESIZED\_WIRE\_11,

Y => SYNTHESIZED\_WIRE\_12);

b2v\_inst18 : hexadisplay

PORT MAP(input\_1 => CLK,

dip\_1 => HEX\_OUT(3 DOWNTO 0),

dip\_2 => HEX\_OUT(7 DOWNTO 4),

dip\_3 => HEX\_OUT(11 DOWNTO 8),

dip\_4 => HEX\_OUT(15 DOWNTO 12),

output\_1 => OUTPUT1,

output\_2 => OUTPUT2,

output\_3 => OUTPUT3,

output\_4 => OUTPUT4,

sal => sal);

b2v\_inst19 : register\_file

PORT MAP(CLK => CLK\_BTN,

RESET => RESET,

RW => control\_word(4),

AA => control\_word(16 DOWNTO 14),

BA => control\_word(13 DOWNTO 11),

D\_data\_in => SYNTHESIZED\_WIRE\_12,

DA => control\_word(19 DOWNTO 17),

REG\_ADD => "001",

A\_data\_out => SYNTHESIZED\_WIRE\_15,

B\_data\_out => SYNTHESIZED\_WIRE\_8,

HEX\_OUT => HEX\_OUT);

b2v\_inst20 : zero\_fill

PORT MAP(z\_in => I(2 DOWNTO 0),

z\_out => SYNTHESIZED\_WIRE\_9);

b2v\_inst8 : branch\_control

PORT MAP(PL => control\_word(2),

JB => control\_word(1),

BC => control\_word(0),

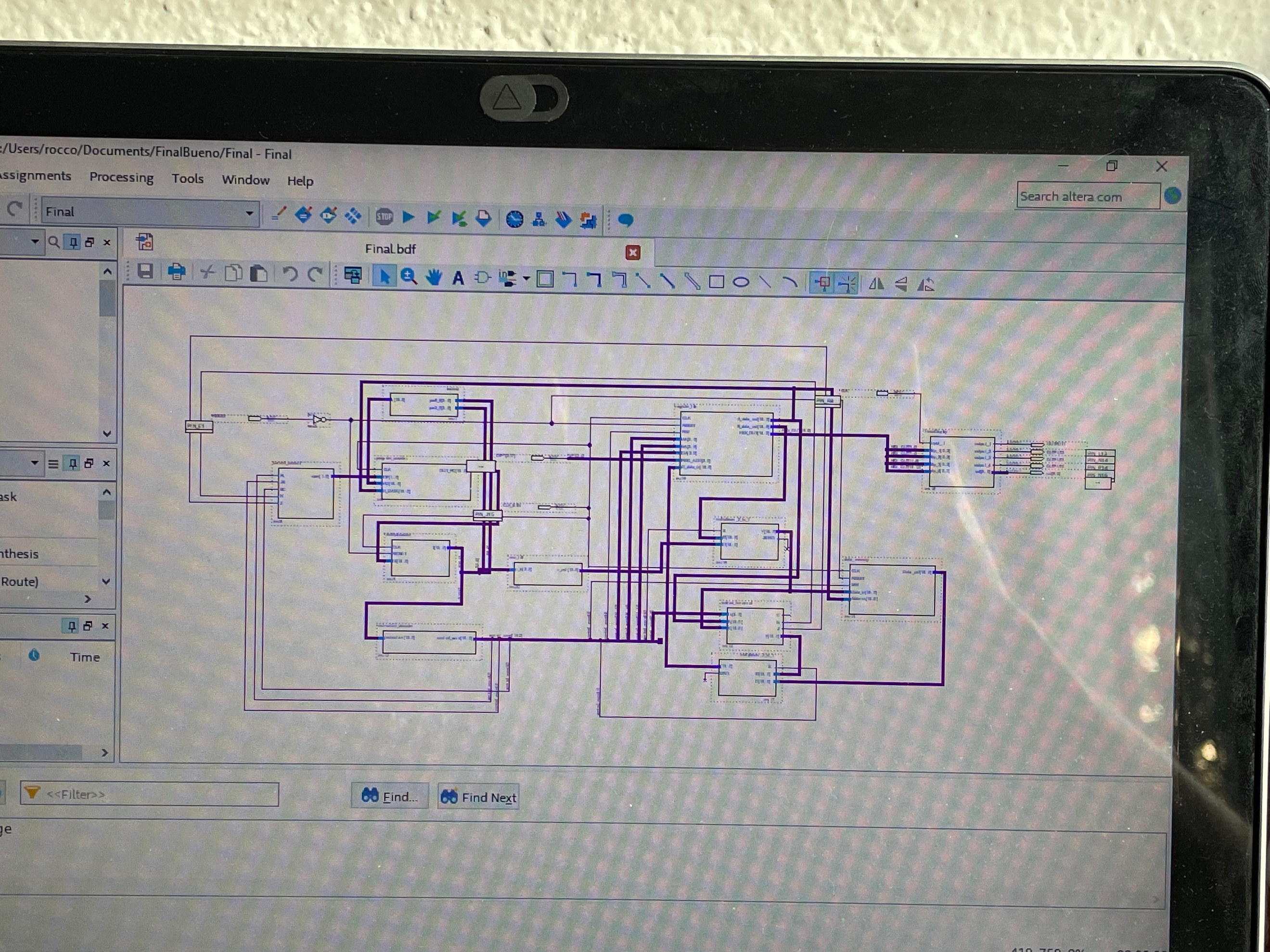
N => SYNTHESIZED\_WIRE\_13,

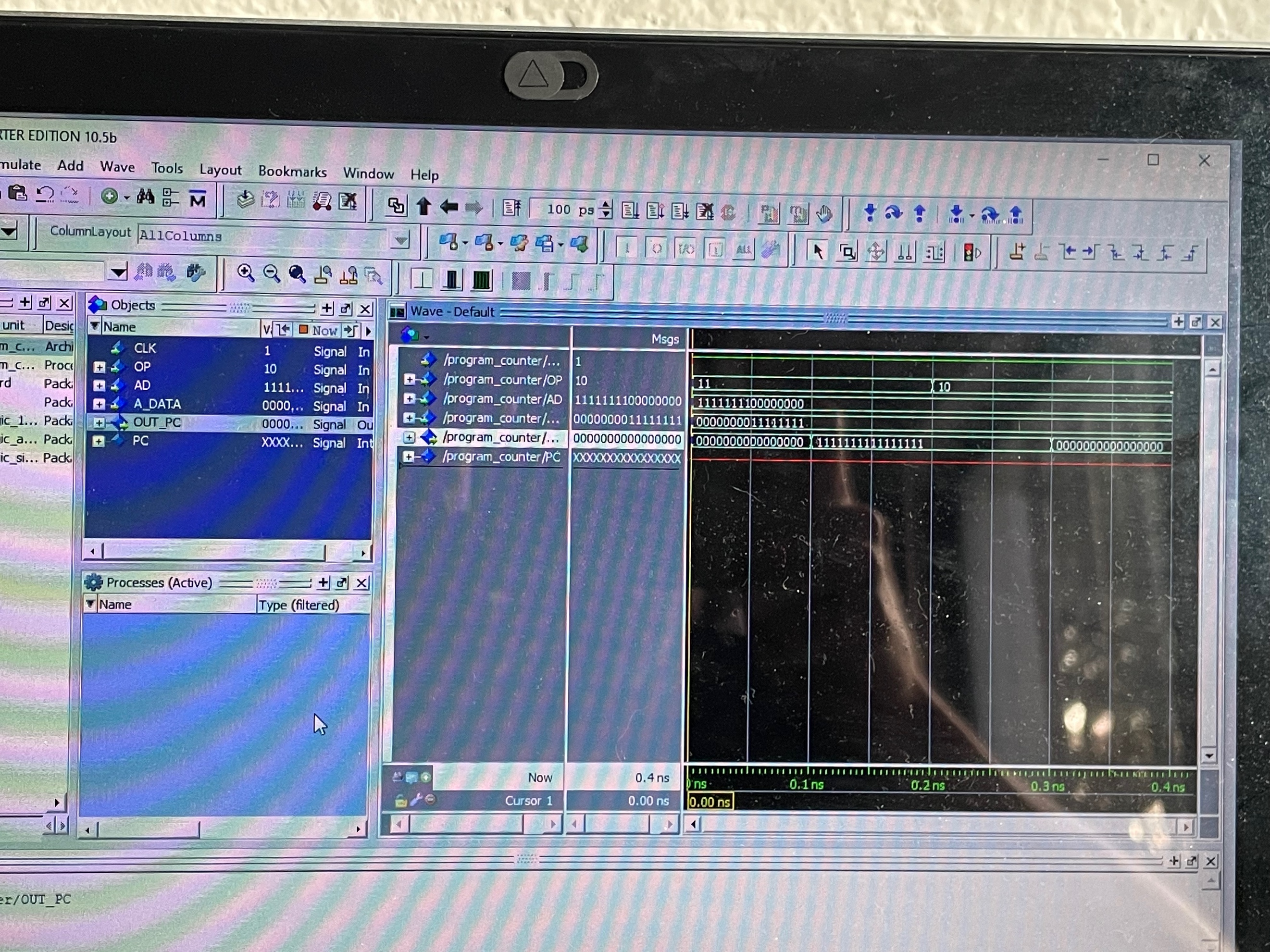
Z => SYNTHESIZED\_WIRE\_14,

oper => SYNTHESIZED\_WIRE\_2);

END bdf\_type;

Para tener al final el esquemático de la siguiente manera:



**Simulación**

**Conclusiones**

Un PC es fundamental para entender el funcionamiento de procesamiento de una computadora y en la formación como ingeniero en computación. Esto, porque en mis propias palabras es cómo la computadora recuerda, registra y da instrucciones en un ciclo secuencial. Es muy interesante que dependiendo del tipo de instrucción se decide ejecutarse o pasar a la siguiente dirección de memoria para ejecutarse en casos más complejos o que no se puede realizar en esa instrucción del ciclo. Como lo dice su nombre, el programa tiene un contador que aumenta automáticamente en cada ciclo de la instrucción, por eso es un circuito secuencial desde la memoria. Cuando una instrucción interrumpe el ciclo, se coloca un nuevo valor en el contador, y el reset lógicamente lo reinicia para comenzar de nuevo. Finalmente, me gustó la complementación de esta última práctica para entender conceptos visto en teoría y también en partes durante prácticas anteriores por lo cual me quedo con satisfacción de terminar el laboratorio de circuitos lógicos.

**Referencias:**

* Prácticas realizadas antes de la final
* Morris, and Michael D. Ciletti. Ed. Pearson, Fifth edition. [3] Digital Design. Principles and Practices. John F. Wakerley. Ed. Pearson.
* Digital Electronics, James Binell and Robert Donovan. Ed. Delmar Publishers.
* Sistemas digitales. Principios y aplicaciones. Ronald J. Tocci. Ed. Pearson Education.
* Logic Design, Wai-Kai Chen. Ed. CRC Press.