Martínez Coronel Brayan Yosafat

# Implementación del Procesador

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use WORK.PAQUETE\_DATOS.ALL;

entity ESCOMIPS is

Port (

CLR, CLK : in STD\_LOGIC;

t\_alu, t\_RR1, t\_RR2, t\_PC, t\_bus\_SR : out STD\_LOGIC\_VECTOR (15 downto 0);

t\_inst : out STD\_LOGIC\_VECTOR (24 downto 0)

);

end ESCOMIPS;

architecture Behavioral of ESCOMIPS is

signal D, fun\_code : STD\_LOGIC\_VECTOR (3 downto 0);

signal op\_code : STD\_LOGIC\_VECTOR (4 downto 0);

signal microinst : STD\_LOGIC\_VECTOR (19 downto 0);

begin

unidad : unidad\_control Port map (

CLK => CLK,

CLR => CLR,

D => D,

sufix => fun\_code,

prefix => op\_code,

microinst => microinst

);

datos : ruta Port map (

clk => CLK,

clr => CLR,

WPC => microinst(16),

UP => microinst(18),

DW => microinst(17),

SHE => microinst(12),

DIR => microinst(11),

WR => microinst(10),

WD => microinst(2),

SR2 => microinst(15),

SWD => microinst(14),

SR => microinst(1),

SDMP => microinst(19),

SDMD => microinst(3),

SOP1 => microinst(9),

SOP2 => microinst(8),

SEXT => microinst(13),

alu\_op => microinst(7 downto 4),

prefix => op\_code,

flags => D,

sufix => fun\_code,

t\_alu => t\_alu,

t\_RR1 => t\_RR1,

t\_RR2 => t\_RR2,

t\_PC => t\_PC,

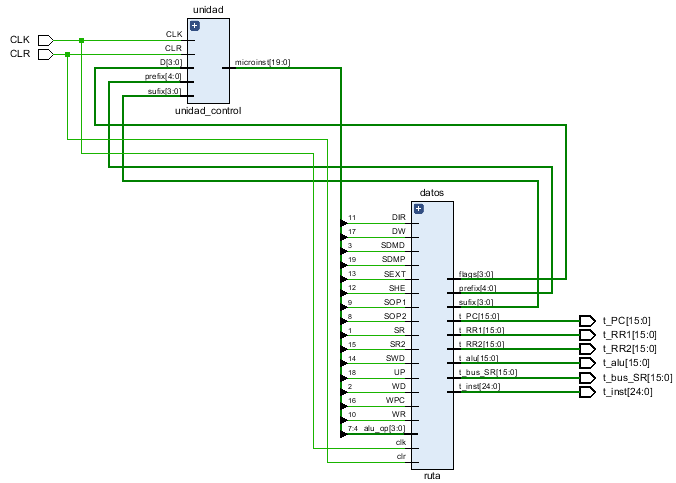
t\_bus\_SR => t\_bus\_SR,

t\_inst => t\_inst

);

end Behavioral;

# Diagrama RTL



# Suma en bucle

## Código de memoria de programa

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity memoria\_programa is

generic (

m : integer := 10;

n : integer := 25

);

Port ( pc : in STD\_LOGIC\_VECTOR (m-1 downto 0);

inst : out STD\_LOGIC\_VECTOR (n-1 downto 0)

);

end entity;

architecture Behavioral of memoria\_programa is

type arreglo is array (0 to (2\*\*m - 1)) of STD\_LOGIC\_VECTOR (n-1 downto 0);

signal mem : arreglo := (others=>(others=>'0'));

constant code\_LI : std\_logic\_vector (4 downto 0) := "00001";

constant code\_ADD : std\_logic\_vector (4 downto 0) := "00000";

constant code\_SWI : std\_logic\_vector (4 downto 0) := "00011";

constant code\_ADDI : std\_logic\_vector (4 downto 0) := "00101";

constant code\_BNEI : std\_logic\_vector (4 downto 0) := "01110";

constant code\_NOP : std\_logic\_vector (4 downto 0) := "10110";

constant code\_B : std\_logic\_vector (4 downto 0) := "10011";

constant fun\_ADD : std\_logic\_vector (3 downto 0) := "0000";

constant SU : std\_logic\_vector (3 downto 0) := "0000";

constant R0 : std\_logic\_vector (3 downto 0) := "0000";

constant R1 : std\_logic\_vector (3 downto 0) := "0000";

begin

mem(0) <= code\_LI & R0 & x"0001"; -- R0 = 1

mem(1) <= code\_LI & R1 & x"0007"; -- R1 = 7

mem(2) <= code\_ADD & R1 & R1 & R0 & SU & fun\_ADD; -- ADD R1 = R1 + R0

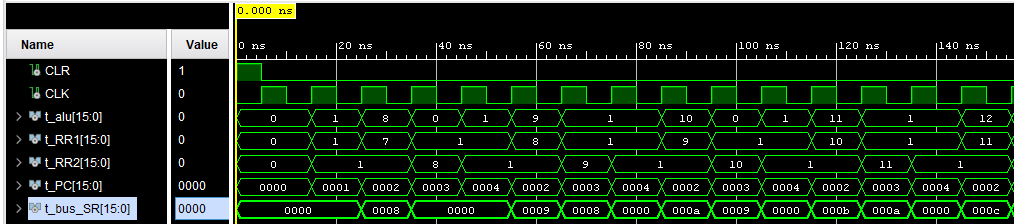
mem(3) <= code\_SWI & R1 & x"0005"; -- Mem[5] = R1

mem(4) <= code\_B & SU & x"0002"; -- B 2

inst <= mem(conv\_integer(pc));

end Behavioral;

## simulación



## Tabla de resultados

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bus | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 |
| PC | 0 | 1 | 2 | 3 | 4 | 2 | 3 | 4 | 2 | 3 | 4 |
| Inst | LI R0 1 | LI R1 7 | ADD R1 R0 | SWI R1 5 | B 2 | ADD R1 R0 | SWI R1 5 | B 2 | ADD R1 R0 | SWI R1 5 | B 2 |
| Read 1 | 0 | 1 | 7 | 1 | 1 | 8 | 1 | 1 | 9 | 1 | 1 |
| Read 2 | 0 | 1 | 1 | 8 | 1 | 1 | 9 | 1 | 1 | 10 | 1 |
| ALU | 0 | 1 | 8 | 0 | 1 | 9 | 1 | 1 | 10 | 0 | 1 |
| BUS SR | 0 | 0 | 8 | 0 | 0 | 9 | 8 | 0 | 10 | 9 | 0 |

# Mínimo del arreglo y Fibonacci de ese número

## Código de memoria de programa

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity memoria\_programa is

generic (

m : integer := 10;

n : integer := 25

);

Port ( pc : in STD\_LOGIC\_VECTOR (m-1 downto 0);

inst : out STD\_LOGIC\_VECTOR (n-1 downto 0)

);

end entity;

architecture Behavioral of memoria\_programa is

type arreglo is array (0 to (2\*\*m - 1)) of STD\_LOGIC\_VECTOR (n-1 downto 0);

signal mem : arreglo := (others=>(others=>'0'));

constant code\_LI : std\_logic\_vector (4 downto 0) := "00001";

constant code\_LWI : std\_logic\_vector (4 downto 0) := "00010";

constant code\_LW : std\_logic\_vector (4 downto 0) := "10111";

constant code\_SW : std\_logic\_vector (4 downto 0) := "00100";

constant code\_SWI : std\_logic\_vector (4 downto 0) := "00011";

constant code\_ADDI : std\_logic\_vector (4 downto 0) := "00101";

constant code\_BGTI : std\_logic\_vector (4 downto 0) := "10001";

constant code\_BLTI : std\_logic\_vector (4 downto 0) := "01111";

constant code\_BLETI : std\_logic\_vector (4 downto 0) := "10000";

constant code\_BNEI : std\_logic\_vector (4 downto 0) := "01110";

constant code\_NOP : std\_logic\_vector (4 downto 0) := "10110";

constant code\_B : std\_logic\_vector (4 downto 0) := "10011";

constant code\_fun : std\_logic\_vector (4 downto 0) := "00000";

constant fun\_ADD : std\_logic\_vector (3 downto 0) := "0000";

constant SU : std\_logic\_vector (3 downto 0) := "0000";

constant R0 : std\_logic\_vector (3 downto 0) := "0000";

constant R1 : std\_logic\_vector (3 downto 0) := "0001";

constant R2 : std\_logic\_vector (3 downto 0) := "0010";

constant R3 : std\_logic\_vector (3 downto 0) := "0011";

constant R4 : std\_logic\_vector (3 downto 0) := "0100";

constant R5 : std\_logic\_vector (3 downto 0) := "0101";

constant R6 : std\_logic\_vector (3 downto 0) := "0110";

constant R7 : std\_logic\_vector (3 downto 0) := "0111";

begin

--Inicializar arreglo

mem(0) <= code\_LI & R0 & x"0000"; -- R0 = 0

mem(1) <= code\_LI & R1 & x"0007"; -- R1 = #

mem(2) <= code\_SW & R1 & R0 & x"000"; -- Mem[R0 + 0] = R1

mem(3) <= code\_LI & R1 & x"0003"; -- R1 = #

mem(4) <= code\_SW & R1 & R0 & x"001"; -- Mem[R0 + 1] = R1

mem(5) <= code\_LI & R1 & x"000A"; -- R1 = #

mem(6) <= code\_SW & R1 & R0 & x"002"; -- Mem[R0 + 2] = R1

mem(7) <= code\_LI & R1 & x"00A0"; -- R1 = #

mem(8) <= code\_SW & R1 & R0 & x"003"; -- Mem[R0 + 3] = R1

mem(9) <= code\_LI & R1 & x"000B"; -- R1 = #

mem(10) <= code\_SW & R1 & R0 & x"004"; -- Mem[R0 + 4] = R1

mem(11) <= code\_LI & R1 & x"0060"; -- R1 = #

mem(12) <= code\_SW & R1 & R0 & x"005"; -- Mem[R0 + 5] = R1

mem(13) <= code\_LI & R1 & x"00B2"; -- R1 = #

mem(14) <= code\_SW & R1 & R0 & x"006"; -- Mem[R0 + 6] = R1

mem(15) <= code\_LI & R1 & x"00A0"; -- R1 = #

mem(16) <= code\_SW & R1 & R0 & x"007"; -- Mem[R0 + 7] = R1

mem(17) <= code\_LI & R1 & x"0A00"; -- R1 = #

mem(18) <= code\_SW & R1 & R0 & x"008"; -- Mem[R0 + 8] = R1

mem(19) <= code\_LI & R1 & x"00C0"; -- R1 = #

mem(20) <= code\_SW & R1 & R0 & x"009"; -- Mem[R0 + 9] = R1

mem(21) <= code\_LI & R1 & x"0017"; -- R1 = #

mem(22) <= code\_SW & R1 & R0 & x"00A"; -- Mem[R0 + 10] = R1

mem(23) <= code\_LI & R1 & x"00B9"; -- R1 = #

mem(24) <= code\_SW & R1 & R0 & x"00B"; -- Mem[R0 + 11] = R1

mem(25) <= code\_LI & R1 & x"0C01"; -- R1 = #

mem(26) <= code\_SW & R1 & R0 & x"00C"; -- Mem[R0 + 12] = R1

mem(27) <= code\_LI & R1 & x"00B2"; -- R1 = #

mem(28) <= code\_SW & R1 & R0 & x"00D"; -- Mem[R0 + 13] = R1

mem(29) <= code\_LI & R1 & x"0003"; -- R1 = #

mem(30) <= code\_SW & R1 & R0 & x"00E"; -- Mem[R0 + 14] = R1

--Obtener el menor del arreglo

mem(31) <= code\_LI & R2 & x"000E"; -- R2 = 14

mem(32) <= code\_LW & R1 & R0 & x"000"; -- R1 = Mem[R0 + 0]

mem(33) <= code\_BGTI & R2 & R0 & x"006"; --if (R0 > R2) goto 33+6

mem(34) <= code\_LW & R3 & R0 & x"000"; -- R3 = Mem[R0 + 0]

mem(35) <= code\_BGTI & R1 & R3 & x"002"; --if (R3 > R1) goto 35+2

mem(36) <= code\_ADDI & R1 & R3 & x"000"; -- ADDI R1 = R3 + 0

mem(37) <= code\_ADDI & R0 & R0 & x"001"; -- ADDI R0 = R0 + 1

mem(38) <= code\_B & SU & x"0021"; -- B 33

--Fibonacci

mem(39) <= code\_LI & R4 & x"0000"; -- R4 = 0

mem(40) <= code\_BLETI & R4 & R1 & x"009"; --if (R1 <= R4) goto 40+9

mem(41) <= code\_LI & R5 & x"0000"; -- R5 = 0

mem(42) <= code\_LI & R6 & x"0001"; -- R6 = 1

mem(43) <= code\_fun & R7 & R5 & R6 & SU & fun\_ADD; -- ADD R7 = R5 + R6

mem(44) <= code\_SW & R7 & R4 & x"00F"; -- Mem[R4 + 15] = R7

mem(45) <= code\_ADDI & R4 & R4 & x"001"; -- ADDI R4 = R4 + 1

mem(46) <= code\_ADDI & R5 & R6 & x"000"; -- ADDI R5 = R6 + 0

mem(47) <= code\_ADDI & R6 & R7 & x"000"; -- ADDI R6 = R7 + 0

mem(48) <= code\_BGTI & R4 & R1 & x"FFB"; --if (R1 > R4) goto 48-5

mem(49) <= code\_NOP & SU & SU & SU & SU & SU;

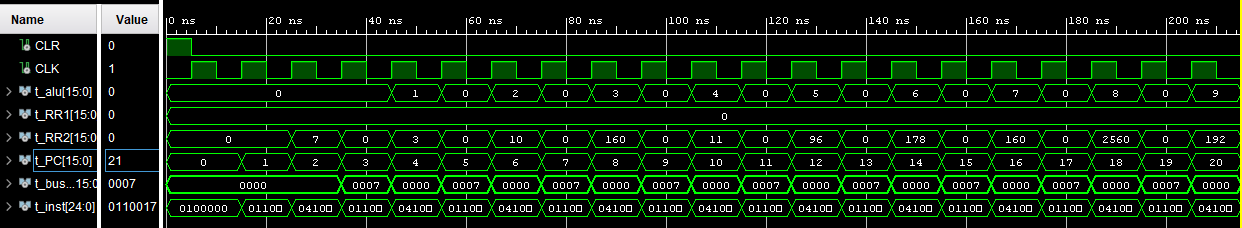
mem(50) <= code\_B & SU & x"0031"; -- B 49

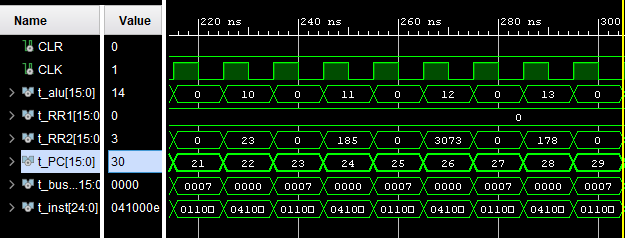
inst <= mem(conv\_integer(pc));

end Behavioral;

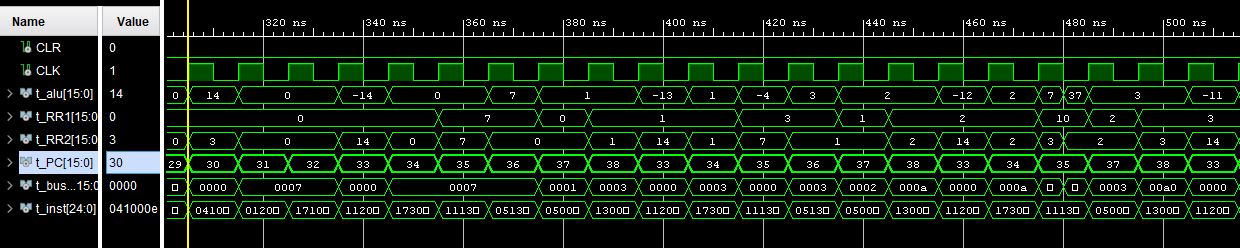
## Simulación

Primero se inicializa el arreglo

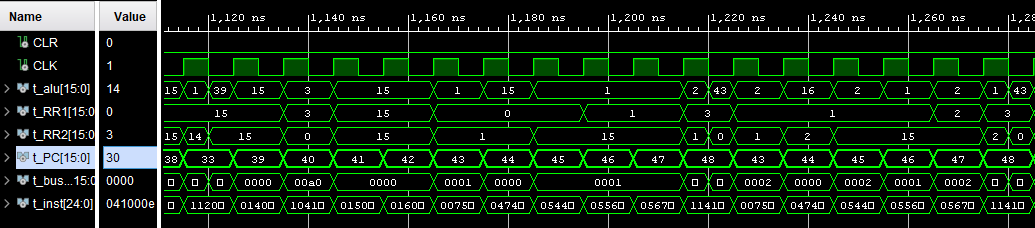




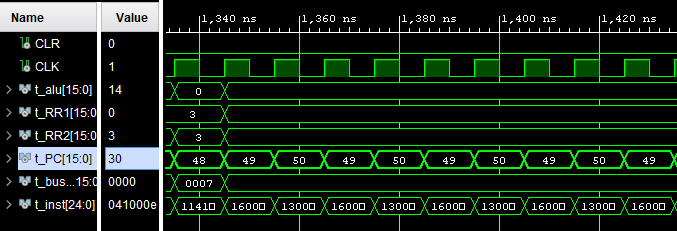
Luego se busca el menor de los números



Entra en bucle 15 veces (el tamaño del arreglo) y obtiene el menor. Ahora se calculan los términos de la serie. Entra en bucle el número menor encontrado.



Al final llega al NOP y B



## Tabla de resultados

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bus | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 |
| PC | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Inst | 0 | LI R0 0 | LI R0 7 | Mem[R0 + 0] = R1 | LI R1 = 3 | Mem[R0 + 0] = R1 | LI R1 = 10 | Mem[R0 + 0] = R1 | LI R1 = 160 | Mem[R0 + 0] = R1 | LI R1 = 11 |
| Read 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Read 2 | 0 | 0 | 7 | 0 | 3 | 0 | 10 | 0 | 160 | 0 | 11 |
| ALU | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 3 | 0 | 4 |
| BUS SR | 0 | 0 | 0 | 7 | 0 | 7 | 0 | 7 | 0 | 7 | 0 |