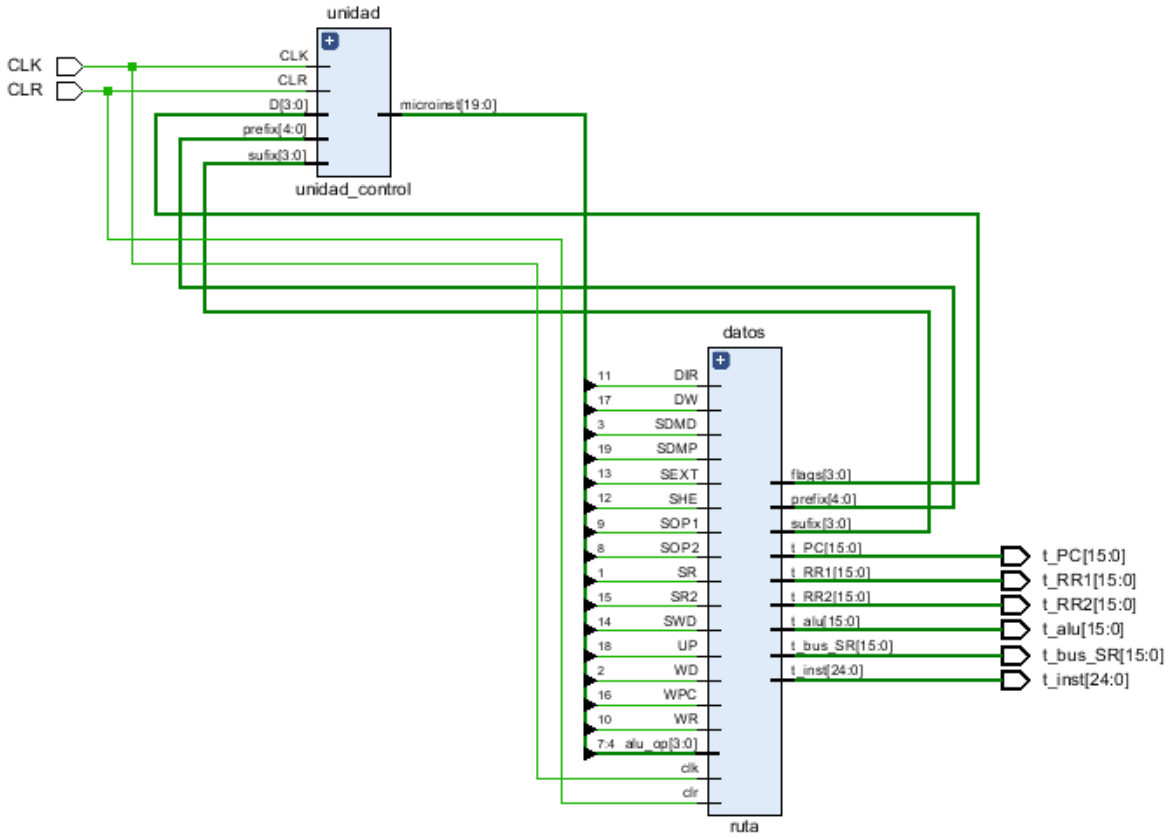
Martínez Coronel Brayan Yosafat

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

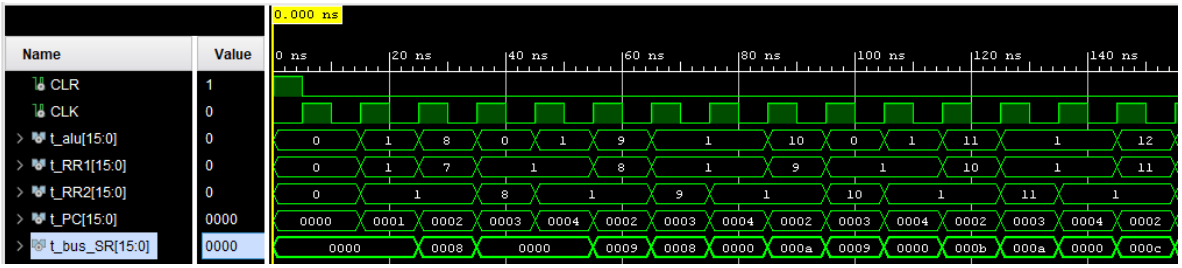


TABLA DE RESULTADOS

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

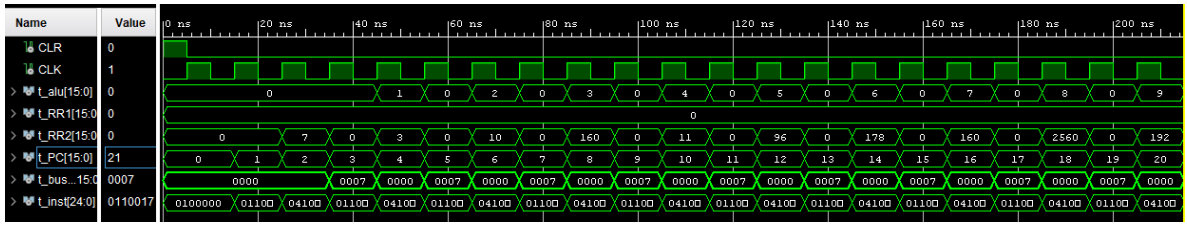
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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";
```

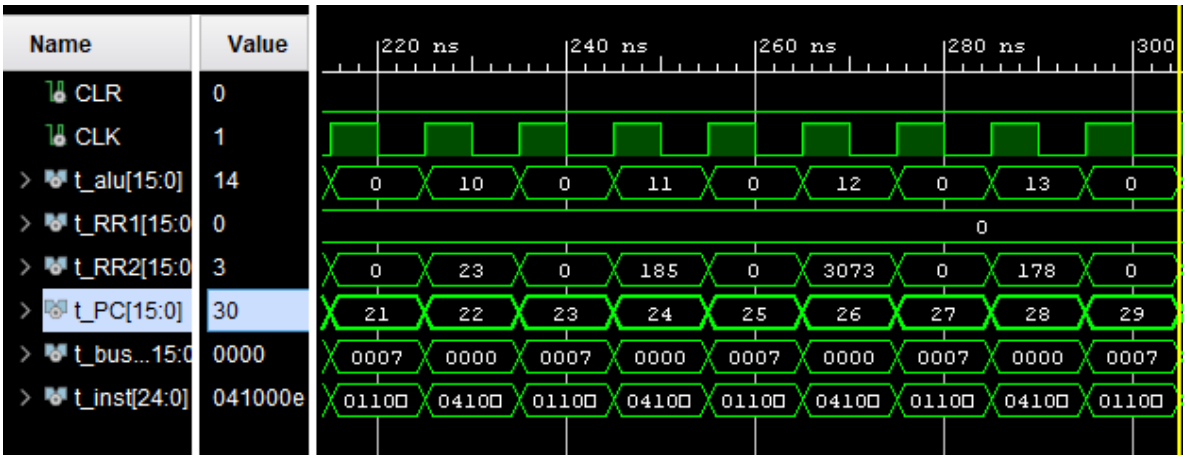
```
constant code_SW : std_logic_vector (4 downto 0) := "00100";
constant code_SWI : std_logic_vector (4 downto 0) := "00011";
constant code_ADDI : std_logic_vector (4 downto 0) := "00101";
constant code_BGTI : std_logic_vector (4 downto 0) := "10001";
constant code_BLTI : std_logic_vector (4 downto 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;
```

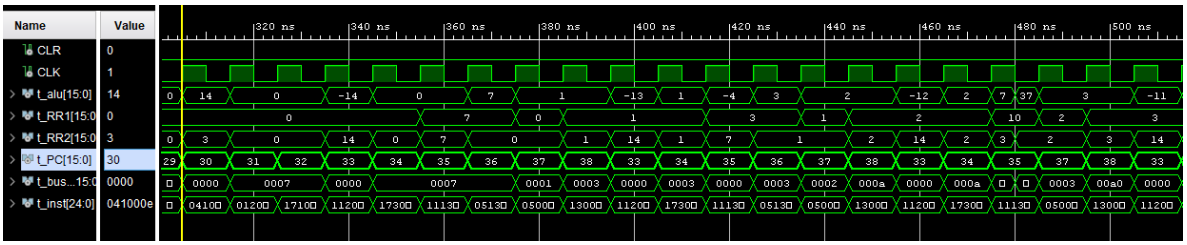
SIMULACIÓN

Primero se inicializa el arreglo

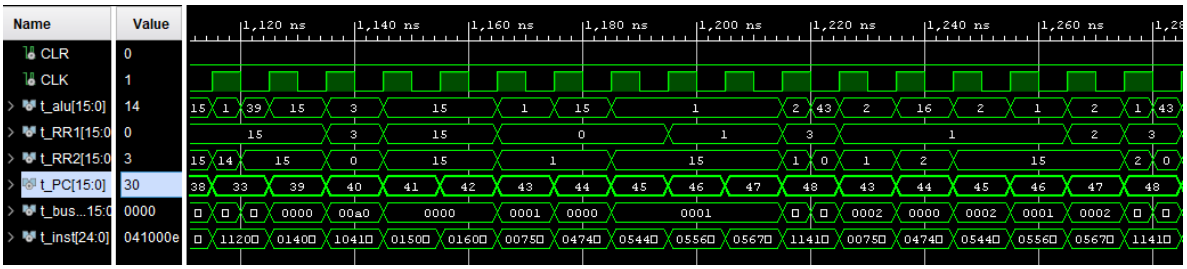




Luego se busca el menor de los números



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

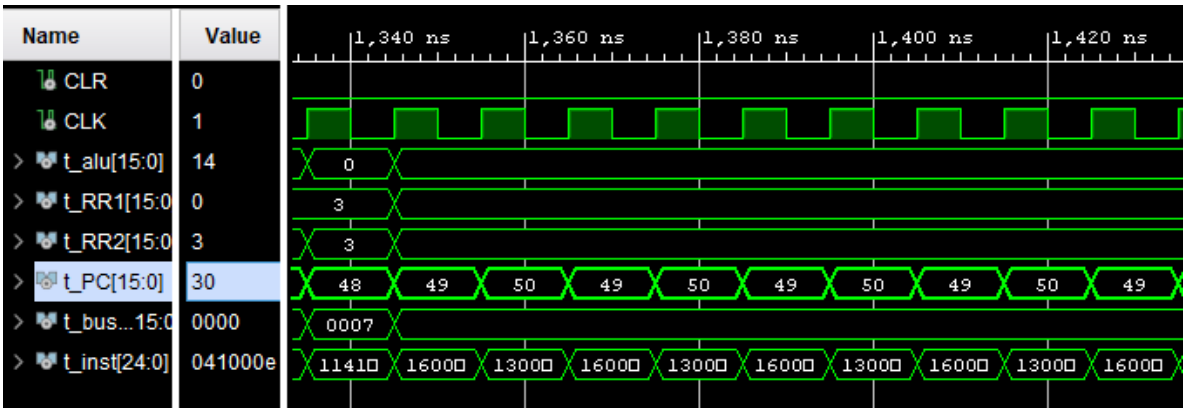


TABLA DE RESULTADOS

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