Martínez Coronel Brayan Yosafat

CÓDIGO DE IMPLEMENTACIÓN

DECODIFICADOR

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

entity decodificador is
    Port (
        id : in STD_LOGIC_VECTOR (4 downto 0);
        TIPOR, BEQI, BNEI, BLTI, BLETI, BGTI, BGETI : out STD_LOGIC
    );
end decodificador;

architecture Behavioral of decodificador is
begin
    with id select
        TIPOR <= '1' when "00000",
                '0' when others;

    with id select
        BEQI <= '1' when "01101",
                '0' when others;

    with id select
        BNEI <= '1' when "01110",
                '0' when others;

    with id select
        BLTI <= '1' when "01111",
                '0' when others;

    with id select
        BLETI <= '1' when "10000",
                '0' when others;

    with id select
        BGTI <= '1' when "10001",
                '0' when others;

    with id select
        BGETI <= '1' when "10010",
                '0' when others;
end Behavioral;
```

MEMORIA DE CÓDIGO DE FUNCIÓN

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

entity memoria_fun_code is
    generic (
        m : integer := 4;
        n : integer := 20
    );
    Port ( index : in STD_LOGIC_VECTOR (m-1 downto 0);
           code : out STD_LOGIC_VECTOR (n-1 downto 0)
    );
end entity;

architecture Behavioral of memoria_fun_code is
type arreglo is array (0 to (2**m - 1)) of STD_LOGIC_VECTOR (n-1 downto 0);
signal mem : arreglo := (others=>(others=>'0'));

begin
    mem(0) <= "00000100010000110011"; --ADD
    mem(1) <= "00000100010001110011"; --SUB
    mem(2) <= "00000100010000000011"; --AND
    mem(3) <= "00000100010000010011"; --OR
    mem(4) <= "00000100010000100011"; --XOR
    mem(5) <= "00000100010011010011"; --NAND
    mem(6) <= "00000100010011000011"; --NOR
    mem(7) <= "00000100010001100011"; --XNOR
    mem(8) <= "00000100010011000011"; --NOT
    mem(9) <= "00000001110000000000"; --SLL
    mem(10) <= "00000001010000000000"; --SRL

    code <= mem(conv_integer(index));
end Behavioral;
```

MEMORIA DE CÓDIGO DE OPERACIÓN

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

entity memoria_op_code is
    generic (
        m : integer := 5;
        n : integer := 20
    );
    Port ( index : in STD_LOGIC_VECTOR (m-1 downto 0);
        code : out STD_LOGIC_VECTOR (n-1 downto 0)
    );
end entity;

architecture Behavioral of memoria_op_code is
type arreglo is array (0 to (2**m - 1)) of STD_LOGIC_VECTOR (n-1 downto 0);
signal mem : arreglo := (others=>(others=>'0'));
begin
    mem(0) <= "00001000000001110001"; --Verificacion
    mem(1) <= "00000000010000000000"; --LI
    mem(2) <= "00000100010000001000"; --LWI
    mem(3) <= "00001000000000001100"; --SWI
    mem(4) <= "00001010000100110101"; --SW
    mem(5) <= "00000100010100110011"; --ADDI
    mem(6) <= "00000100010101110011"; --SUBI
    mem(7) <= "00000110010100000011"; --ANDI
    mem(8) <= "00000110010100010011"; --ORI
    mem(9) <= "00000100010100100011"; --XORI
    mem(10) <= "00000100010111010011"; --NANDI
    mem(11) <= "00000100010111000011"; --NORI
    mem(12) <= "00000100010101100011"; --XNORI

    mem(13) <= "10010000001100110011"; --BEQI
    mem(14) <= "10010000001100110011"; --BNEI
    mem(15) <= "10010000001100110011"; --BLTI
    mem(16) <= "10010000001100110011"; --BLETI
    mem(17) <= "10010000001100110011"; --BGTI
    mem(18) <= "10010000001100110011"; --BGETI

    mem(19) <= "00010000000000000000"; --B
    mem(20) <= "01010000000000000000"; --CALL
    mem(21) <= "00100000000000000000"; --RET
    mem(22) <= "00000000000000000000"; --NOP
    mem(23) <= "00000110010100110001"; --LW

    code <= mem(conv_integer(index));
end Behavioral;
```

NIVEL

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

entity nivel is
    Port (
        CLK, CLR : in STD_LOGIC;
        NA : out STD_LOGIC
    );
end nivel;
```

```
architecture Behavioral of nivel is
signal up_clk, down_clk : STD_LOGIC := '0';
begin
    alto: process (CLK, CLR)
    begin
        if (CLR = '1') then
            up_clk <= '0';
        elsif (rising_edge(CLK)) then
            up_clk <= not up_clk;
        end if;
    end process;

    bajo: process (CLK, CLR, up_clk)
    begin
        if (CLR = '1') then
            down_clk <= '0';
        elsif (falling_edge(CLK)) then
            down_clk <= not down_clk;
        end if;
    end process;

    NA <= up_clk xor down_clk;
end Behavioral;
```

CONDICIÓN

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

entity condiciones is
    Port (
        Q : in STD_LOGIC_VECTOR (3 downto 0);
        EQ, NE, LT, LE, GT, GE : out STD_LOGIC
    );
end condiciones;

architecture Behavioral of condiciones is
begin
    -- Q = "OV N Z C"
    EQ <= Q(1);
    NE <= not Q(1);
    LT <= not Q(0);
    LE <= Q(1) or not Q(0);
    GT <= not Q(1) and Q(0);
    GE <= Q(0);
end Behavioral;
```

REGISTRO DE BANDERAS

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

entity banderas is
    Port (
        LF, CLK, CLR : in STD_LOGIC;
        D : in STD_LOGIC_VECTOR (3 downto 0);
        Q : out STD_LOGIC_VECTOR (3 downto 0)
    );
end banderas;

architecture Behavioral of banderas is
signal reg : STD_LOGIC_VECTOR (3 downto 0);
begin
    process (CLK, CLR, D)
    begin
        if (CLR = '1') then
            reg <= "0000";
        elsif falling_edge(CLK) then
            if (LF = '1') then
                reg <= D;
            else
                reg <= "0000";
            end if;
        end if;
    end process;

    Q <= reg;
end Behavioral;
```

CONTROL

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

entity control is
    Port (
        CLK, CLR : in STD_LOGIC;
        EQ, NE, LT, LE, GT, GE, NA : in STD_LOGIC;
        TIPOR, BEQI, BNEI, BLTI, BLETI, BGTI, BGETI : in STD_LOGIC;
        SDOPC, SM : out STD_LOGIC
    );
end control;

architecture Behavioral of control is
type estado is (E0);
signal actual, sig : estado;
begin
```

```
--Para el registro del estado
        process (CLK, CLR)
        begin
                if CLR = '1' then
                        actual <= E0;
                elsif rising_edge(CLK) then
                        actual <= sig;
                end if;
        end process;

process (actual, NA)
begin
    SDOPC <= '0';
    SM <= '1';

    case actual is
    when E0 =>
        if TIPOR = '1' then
            SM <= '0';
        elsif BEQI = '1' then
            if NA = '0' then
                SDOPC <= '0';
            else
                if EQ = '1' then
                    SDOPC <= '1';
                else
                    SDOPC <= '0';
                end if;
            end if;
        elsif BNEI = '1' then
            if NA = '0' then
                SDOPC <= '0';
            else
                if NE = '1' then
                    SDOPC <= '1';
                else
                    SDOPC <= '0';
                end if;
            end if;
        elsif BLTI = '1' then
            if NA = '0' then
                SDOPC <= '0';
            else
                if LT = '1' then
                    SDOPC <= '1';
                else
                    SDOPC <= '0';
                end if;
            end if;
            elsif BLETI = '1' then
                if NA = '0' then
                    SDOPC <= '0';
                else
                    if LE = '1' then
                        SDOPC <= '1';
                    else
                        SDOPC <= '0';
                    end if;
                end if;
            elsif BGTI = '1' then
                if NA = '0' then
                    SDOPC <= '0';
                else
                    if GT = '1' then
                        SDOPC <= '1';
                    else
                        SDOPC <= '0';
                    end if;
                end if;
            elsif BGETI = '1' then
                if NA = '0' then
                    SDOPC <= '0';
                else
                    if GE = '1' then
                        SDOPC <= '1';
                    else
                        SDOPC <= '0';
                    end if;
                end if;
            else
                SDOPC <= '1';
            end if;
        end case;

        sig <= E0;
    end process;
end Behavioral;
```

UNIDAD DE CONTROL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use WORK.PAQUETE_COMPONENTES.ALL;

entity unidad_control is
    Port (
        CLK, CLR : in STD_LOGIC;
        D, sufix : in STD_LOGIC_VECTOR (3 downto 0);
        prefix : in STD_LOGIC_VECTOR (4 downto 0);
        microinst : out STD_LOGIC_VECTOR (19 downto 0)
    );
end unidad_control;

architecture Behavioral of unidad_control is
signal index_op : STD_LOGIC_VECTOR (4 downto 0);
signal Q : STD_LOGIC_VECTOR (3 downto 0);
signal out_op, out_fun, output : STD_LOGIC_VECTOR (19 downto 0);
signal s_TIPOR, s_BEQI, s_BNEI, s_BLTI, s_BLETI, s_BGTI, s_BGETI : STD_LOGIC;
signal s_EQ, s_NE, s_LT, s_LE, s_GT, s_GE, SDOPC, SM, NA : STD_LOGIC;
begin
    t_mem_op : memoria_op_code Port map (
        index => index_op,
        code => out_op
    );

    t_mem_fun : memoria_fun_code Port map (
        index => sufix,
        code => out_fun
    );

    t_deco : decodificador Port map (
        id => prefix,
        TIPOR => s_TIPOR,
        BEQI => s_BEQI,
        BNEI => s_BNEI,
        BLTI => s_BLTI,
        BLETI => s_BLETI,
        BGTI => s_BGTI,
        BGETI => s_BGETI
    );

    t_banderas : banderas Port map (
        LF => output(0),
        CLK => CLK,
        CLR => CLR,
        D => D,
        Q => Q
    );

    t_condiciones : condiciones Port map (
        Q => Q,
        EQ => s_EQ,
        NE => s_NE,
        LT => s_LT,
        LE => s_LE,
        GT => s_GT,
        GE => s_GE
    );

    t_control : control Port map (
        CLK => CLK,
        CLR => CLR,
        EQ => s_EQ,
        NE => s_NE,
        LT => s_LT,
        LE => s_LE,
        GT => s_GT,
        GE => s_GE,
        NA => NA,
        TIPOR => s_TIPOR,
        BEQI => s_BEQI,
        BNEI => s_BNEI,
        BLTI => s_BLTI,
        BLETI => s_BLETI,
        BGTI => s_BGTI,
        BGETI => s_BGETI,
        SDOPC => SDOPC,
        SM => SM
    );

    t_nivel : nivel Port map (
        CLK => CLK,
        CLR => CLR,
        NA => NA
    );

    with SDOPC select
        index_op <= "00000" when '0',
                    prefix when others;

    with SM select
        output <= out_fun when '0',
                    out_op when others;

    microinst <= output;
end Behavioral;
```

CÓDIGO DE SIMULACIÓN

DECODIFICADOR

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

entity tb_deco is
--  Port ( );
```

```
end tb_deco;

architecture Behavioral of tb_deco is
    component decodificador is
        Port (
            id : in STD_LOGIC_VECTOR (4 downto 0);
            TIPOR, BEQI, BNEI, BLTI, BLETI, BGTI, BGETI : out STD_LOGIC
        );
    end component;

    signal opCode : STD_LOGIC_VECTOR (4 downto 0);
    signal TIPOR, BEQI, BNEQI, BLTI, BLETI, BGTI, BGETI : STD_LOGIC;

begin
    deco : decodificador Port map (
        id => opCode,
        TIPOR => TIPOR,
        BEQI => BEQI,
        BNEI => BNEQI,
        BLTI => BLTI,
        BLETI => BLETI,
        BGTI => BGTI,
        BGETI => BGETI
    );

    process
    begin
        opCode <= "00000";
        wait for 10 ns;
        opCode <= "01101";
        wait for 10 ns;
        opCode <= "01110";
        wait for 10 ns;
        opCode <= "01111";
        wait for 10 ns;
        opCode <= "10000";
        wait for 10 ns;
        opCode <= "10001";
        wait for 10 ns;
        opCode <= "10010";
        wait;
    end process;
end Behavioral;
```

MEMORIA DE CÓDIGO DE FUNCIÓN

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

entity tb_fun_code is
--  Port ( );
end tb_fun_code;

architecture Behavioral of tb_fun_code is
    component memoria_fun_code is
        generic (
            m : integer := 4;
            n : integer := 20
        );
        Port ( index : in STD_LOGIC_VECTOR (m-1 downto 0);
               code : out STD_LOGIC_VECTOR (n-1 downto 0)
        );
    end component;

    signal funCode : STD_LOGIC_VECTOR (3 downto 0);
    signal microFuncion : STD_LOGIC_VECTOR (19 downto 0);

begin
    fun : memoria_fun_code Port map (
        index => funCode,
        code => microFuncion
    );

    process
    begin
        funCode <= "0000";
        wait for 10 ns;
        funCode <= "0001";
        wait for 10 ns;
        funCode <= "0010";
        wait for 10 ns;
        funCode <= "0011";
        wait for 10 ns;
        funCode <= "0100";
        wait for 10 ns;
        funCode <= "0101";
        wait for 10 ns;
        funCode <= "0110";
        wait for 10 ns;
        funCode <= "0111";
        wait for 10 ns;
        funCode <= "1000";
        wait;
    end process;
end Behavioral;
```

MEMORIA DE CÓDIGO DE OPERACIÓN

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

entity tb_op_code is
--  Port ( );
end tb_op_code;

architecture Behavioral of tb_op_code is
    component memoria_op_code is
        generic (
            m : integer := 5;
            n : integer := 20
        );
        Port ( index : in STD_LOGIC_VECTOR (m-1 downto 0);
               code : out STD_LOGIC_VECTOR (n-1 downto 0)
        );
    end component;

    signal opCode : STD_LOGIC_VECTOR (4 downto 0);
    signal microOpCode : STD_LOGIC_VECTOR (19 downto 0);
begin
    op : memoria_op_code Port map (
        index => OpCode,
        code => microOpCode
    );

    process
    begin
        opCode <= "00000";
        wait for 10 ns;
        opCode <= "00001";
        wait for 10 ns;
        opCode <= "00010";
        wait for 10 ns;
        opCode <= "00011";
        wait for 10 ns;
        opCode <= "00100";
        wait for 10 ns;
        opCode <= "00101";
        wait for 10 ns;
        opCode <= "00110";
        wait for 10 ns;
        opCode <= "00111";
        wait for 10 ns;
        opCode <= "01000";
        wait;
    end process;
end Behavioral;
```

NIVEL

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

entity tb_nivel is
--  Port ( );
end tb_nivel;

architecture Behavioral of tb_nivel is
    component nivel is
        Port (
            CLK, CLR : in STD_LOGIC;
            NA : out STD_LOGIC
        );
    end component;

    signal clr, clk, na : STD_LOGIC;

begin
    niv : nivel Port map (
        clk => clk,
        clr => clr,
        na => na
    );

    reloj : process begin
        clk <= '0';
        wait for 5 ns;
        clk <= '1';
        wait for 5 ns;
    end process;

    process
    begin
        clr <= '1';
        wait for 3 ns;
        clr <= '0';
        wait;
    end process;
end Behavioral;
```

CONDICIÓN

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

entity tb_condiciones is
--  Port ( );
end tb_condiciones;

architecture Behavioral of tb_condiciones is
    component condiciones is
        Port (
            Q : in STD_LOGIC_VECTOR (3 downto 0);
            EQ, NE, LT, LE, GT, GE : out STD_LOGIC
        );
    end component;

    signal EQ, NE, LT, LE, GT, GE : STD_LOGIC;
    signal Q : STD_LOGIC_VECTOR (3 downto 0);

begin
    con : condiciones Port map (
        Q => Q,
        EQ => EQ,
        NE => NE,
        LT => LT,
        LE => LE,
        GT => GT,
        GE => GE
    );

    --Q[OV, N, Z, C]
    process
    begin
        Q <= "0010";
        wait for 10 ns;
        Q <= "1101";
        wait for 10 ns;
        Q <= "0001";
        wait for 10 ns;
        Q <= "0000";
        wait for 10 ns;
        Q <= "1111";
        wait;
    end process;
end Behavioral;
```

REGISTRO DE BANDERAS

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

entity tb_banderas is
--  Port ( );
end tb_banderas;

architecture Behavioral of tb_banderas is
    component banderas is
        Port (
            LF, CLK, CLR : in STD_LOGIC;
            D : in STD_LOGIC_VECTOR (3 downto 0);
            Q : out STD_LOGIC_VECTOR (3 downto 0)
        );
    end component;

    signal clr, clk, LF : STD_LOGIC;
    signal D, Q : STD_LOGIC_VECTOR (3 downto 0);
begin
    reg : banderas port map(
        D => D,
        clr => clr,
        clk => clk,
        LF => LF,
        Q => Q
    );

    reloj : process begin
        clk <= '0';
        wait for 5 ns;
        clk <= '1';
        wait for 5 ns;
    end process;

    process
    begin
        clr <= '1';
        wait until falling_edge(clk);
        clr <= '0';
        LF <= '1';
        D <= "1111";
        wait until falling_edge(clk);
        D <= "0000";
        wait until falling_edge(clk);
        D <= "0001";
        wait until falling_edge(clk);
        D <= "0010";
        wait until falling_edge(clk);
        D <= "0011";
        wait until falling_edge(clk);
        LF <= '0';
        D <= "0000";
        wait until falling_edge(clk);
        D <= "0001";
        wait;
    end process;
end Behavioral;
```

CONTROL

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

entity tb_control is
--  Port ( );
end tb_control;

architecture Behavioral of tb_control is
    component control is
        Port (
            CLK, CLR : in STD_LOGIC;
            EQ, NE, LT, LE, GT, GE, NA : in STD_LOGIC;
            TIPOR, BEQI, BNEI, BLTI, BLETI, BGTI, BGETI : in STD_LOGIC;
            SDOPC, SM : out STD_LOGIC
        );
    end component;

    signal clk, clr, EQ, NE, LT, LE, GT, GE, nivel, BGETI, BGTI, BLETI, BLTI, BNEQI, BEQI, TIPOR, SDOPC, SM : STD_LOGIC := '0';

begin
     uc : control Port map (
        clk => clk,
        clr => clr,
        EQ => EQ,
        NE => NE,
        LT => LT,
        LE => LE,
        GT => GT,
        GE => GE,
        na => nivel,
        BGETI => BGETI,
        BGTI => BGTI,
        BLETI => BLETI,
        BLTI => BLTI,
        BNEI => BNEQI,
        BEQI => BEQI,
        TIPOR => TIPOR,
        SDOPC => SDOPC,
        SM => SM
    );

    reloj : process begin
        clk <= '0';
        wait for 5 ns;
        clk <= '1';
        wait for 5 ns;
    end process;

    process
    begin
        clr <= '1';
        wait until rising_edge(clk);

        clr <= '0';
        TIPOR <= '1';
        wait until rising_edge(clk);

        TIPOR <= '0';
        LT  <= '1';
        BLTI <= '1';
        wait until rising_edge(clk);

        LT <= '0';
        wait until rising_edge(clk);

        BLTI <= '0';
        LE <= '1';
        BLETI <= '1';
        wait until rising_edge(clk);

        LE <= '0';
        wait until rising_edge(clk);

        clr <= '1';
        wait;
    end process;
end Behavioral;
```

UNIDAD DE CONTROL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use STD.TEXTIO.ALL;
use IEEE.std_logic_TEXTIO.ALL;
```

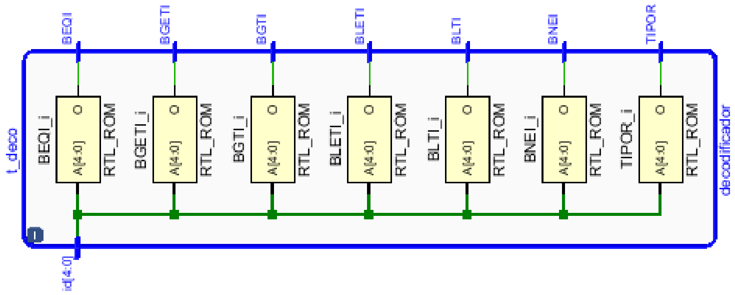
```vhdl
use IEEE.STD_LOGIC_1164.ALL;

entity tb_unidad_control is
--  Port ( );
end tb_unidad_control;

architecture Behavioral of tb_unidad_control is
constant clk_period : time := 10 ns;

component unidad_control is
    Port (
        CLK, CLR, LF : in STD_LOGIC;
        D, sufix : in STD_LOGIC_VECTOR (3 downto 0);
        prefix : in STD_LOGIC_VECTOR (4 downto 0);
        microinst : out STD_LOGIC_VECTOR (19 downto 0)
    );
end component;

signal CLK, CLR, LF : STD_LOGIC;
signal D, sufix : STD_LOGIC_VECTOR (3 downto 0);
signal prefix : STD_LOGIC_VECTOR (4 downto 0);
signal microinst : STD_LOGIC_VECTOR (19 downto 0);
begin
    t_unidad : unidad_control port map (
        CLK => CLK,
        CLR => CLR,
        LF =>LF,
        D => D,
        sufix => sufix,
        prefix => prefix,
        microinst => microinst
    );

    clk_process : process
    begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;

    sim_proc: process
    constant ALTO : string (1 to 4) := "ALTO";
    constant BAJO : string (1 to 4) := "BAJO";
    file ARCH_SALIDA : TEXT;

    variable LINEA_SALIDA : line;

    file ARCH_ENTRADA : TEXT;
```

```
variable LINEA_ENTRADA : line;
variable CADENA_1 : string (1 to 8);
variable CADENA_2 : string (1 to 20);
variable CADENA_3 : string (1 to 5);

variable v_CLK, v_CLR, v_LF : STD_LOGIC;
variable v_D, v_sufix : STD_LOGIC_VECTOR (3 downto 0);
variable v_prefix : STD_LOGIC_VECTOR (4 downto 0);
variable v_microinst : STD_LOGIC_VECTOR (19 downto 0);
begin
    file_open(ARCH_ENTRADA, "C:\Users\yosaf\Desktop\Sexto\Arquitectura\Practica11\entradas.txt", READ_MODE);
    file_open(ARCH_SALIDA, "C:\Users\yosaf\Desktop\Sexto\Arquitectura\Practica11\salidas.txt", WRITE_MODE);
    readline(ARCH_ENTRADA, LINEA_ENTRADA); -- salta la primera linea

    CADENA_1 := " OP_CODE";
    write(LINEA_SALIDA, CADENA_1, right, CADENA_1'LENGTH + 1);
    CADENA_1 := "FUN_CODE";
    write(LINEA_SALIDA, CADENA_1, right, CADENA_1'LENGTH + 1);
    CADENA_1 := "BANDERAS";
    write(LINEA_SALIDA, CADENA_1, right, CADENA_1'LENGTH + 1);
    CADENA_3 := "  CLR";
    write(LINEA_SALIDA, CADENA_3, right, CADENA_3'LENGTH + 1);
    CADENA_3 := "   LF";
    write(LINEA_SALIDA, CADENA_3, right, CADENA_3'LENGTH + 1);
    CADENA_2 := "    MICROINSTRUCCION";
    write(LINEA_SALIDA, CADENA_2, right, CADENA_2'LENGTH + 1);
    CADENA_3 := "NIVEL";
    write(LINEA_SALIDA, CADENA_3, right, CADENA_3'LENGTH + 1);

    writeline(ARCH_SALIDA, LINEA_SALIDA);-- escribe la linea en el archivo

    FOR I IN 1 TO 52 LOOP
        readline(ARCH_ENTRADA, LINEA_ENTRADA); -- lee una linea completa

        read(LINEA_ENTRADA, v_prefix);
        prefix <= v_prefix;
        read(LINEA_ENTRADA, v_sufix);
        sufix <= v_sufix;
        read(LINEA_ENTRADA, v_D);
        D <= v_D;
        read(LINEA_ENTRADA, v_CLR);
        CLR <= v_CLR;
        read(LINEA_ENTRADA, v_LF);
        LF <= v_LF;

        WAIT UNTIL RISING_EDGE(CLK);            --ESPERO AL FLANCO

        v_microinst := microinst;
        write(LINEA_SALIDA, v_prefix, right, 8);
```

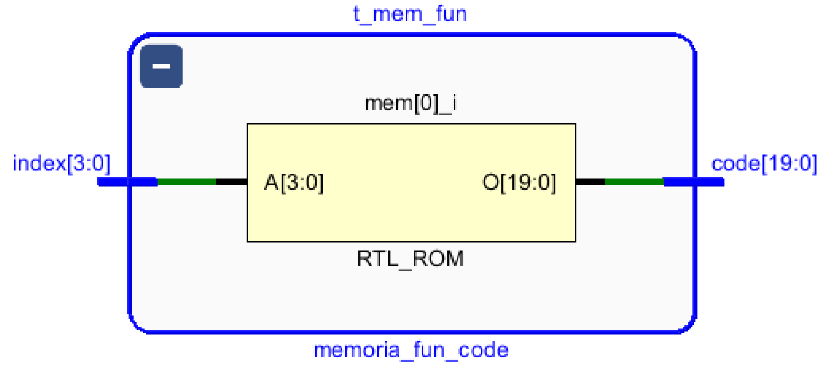
```
      write(LINEA_SALIDA, v_sufix, right, 9);
      write(LINEA_SALIDA, v_D, right, 9);
      write(LINEA_SALIDA, v_CLR, right, 6);
      write(LINEA_SALIDA, v_LF, right, 6);
      write(LINEA_SALIDA, v_microinst, right, 22);
      write(LINEA_SALIDA, ALTO, right, 6);
      writeline(ARCH_SALIDA, LINEA_SALIDA);

      WAIT UNTIL FALLING_EDGE(CLK);

      v_microinst := microinst;
      write(LINEA_SALIDA, v_prefix, right, 8);
      write(LINEA_SALIDA, v_sufix, right, 9);
      write(LINEA_SALIDA, v_D, right, 9);
      write(LINEA_SALIDA, v_CLR, right, 6);
      write(LINEA_SALIDA, v_LF, right, 6);
      write(LINEA_SALIDA, v_microinst, right, 22);
      write(LINEA_SALIDA, BAJO, right, 6);
      writeline(ARCH_SALIDA, LINEA_SALIDA);

    end loop;
    file_close(ARCH_ENTRADA);  -- cierra el archivo
    file_close(ARCH_SALIDA);  -- cierra el archivo

   wait;
  end process;
end Behavioral;
```

DIAGRAMAS RTL

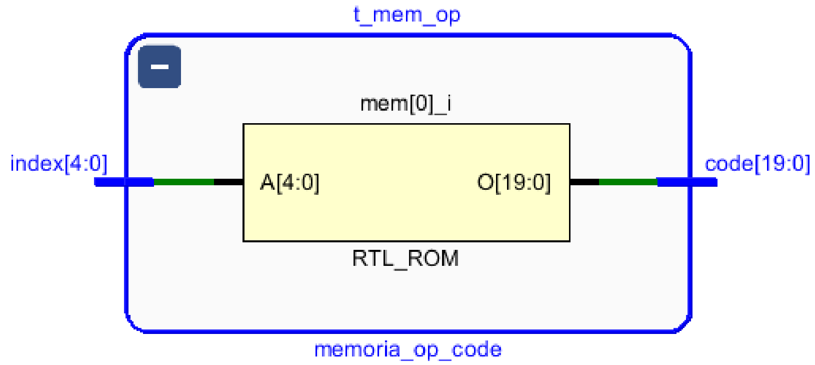
DECODIFICADOR



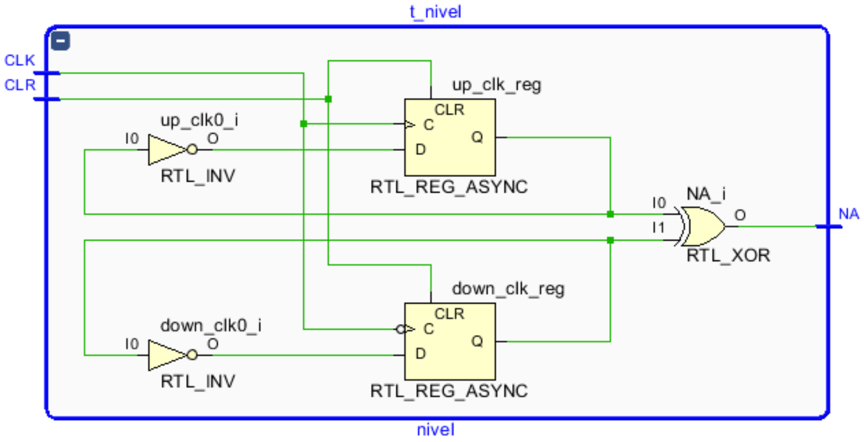
MEMORIA DE CÓDIGO DE FUNCIÓN



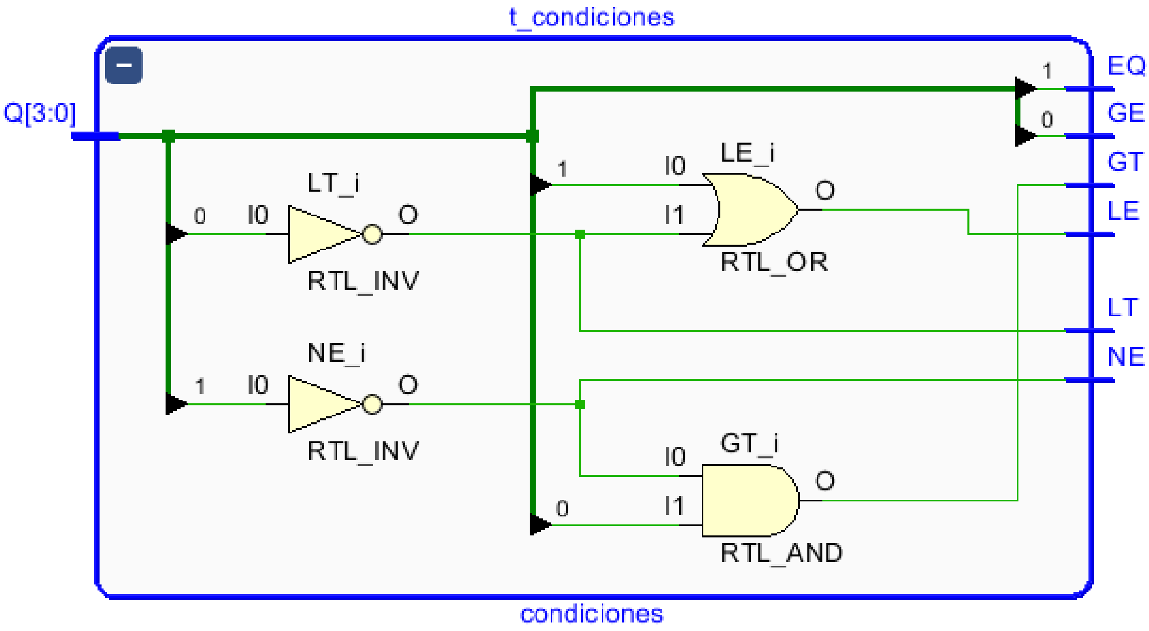
MEMORIA DE CÓDIGO DE OPERACIÓN



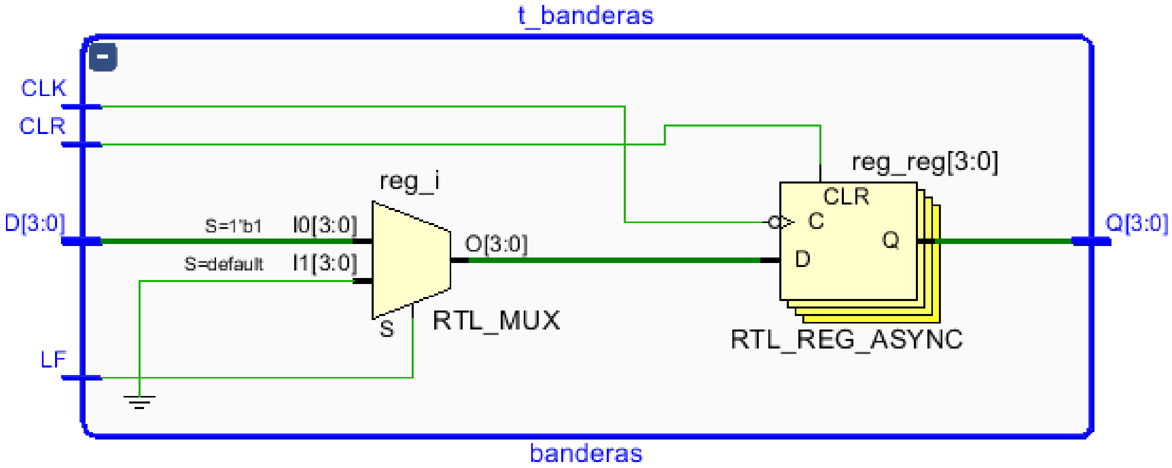
NIVEL



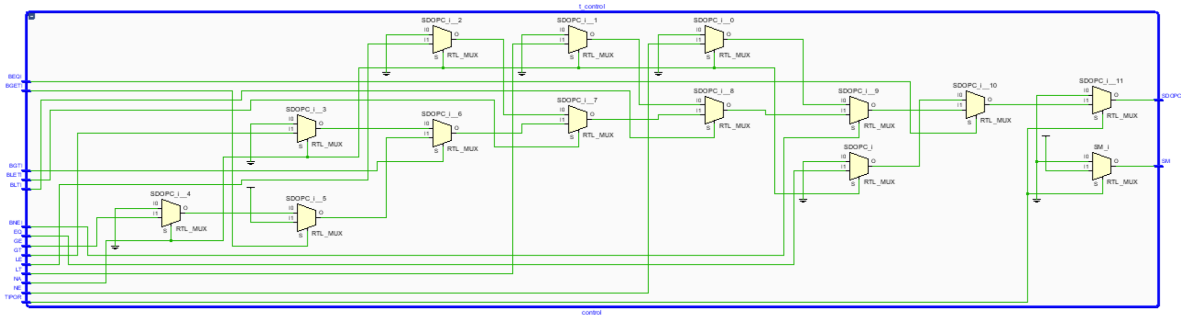
CONDICIÓN



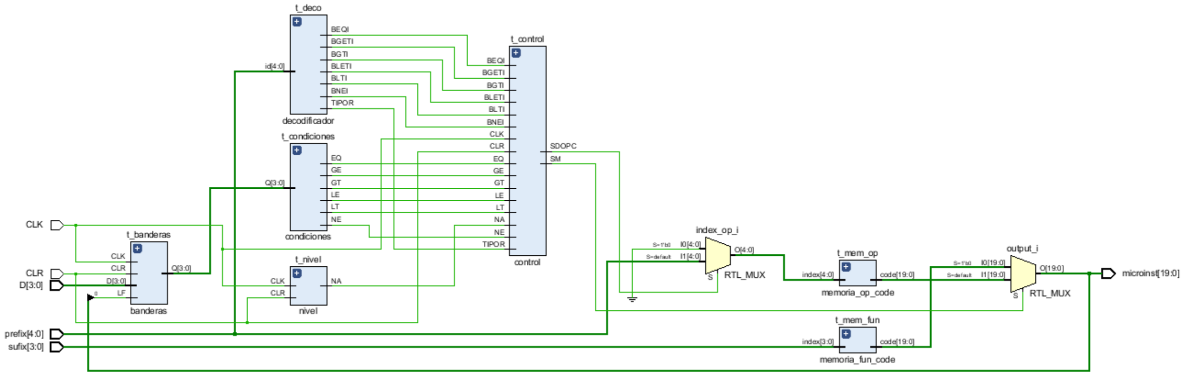
REGISTRO DE BANDERAS



CONTROL

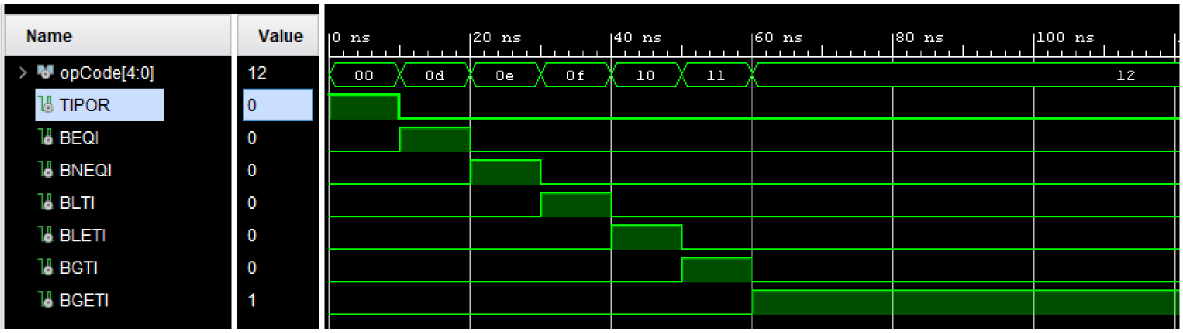


UNIDAD DE CONTROL



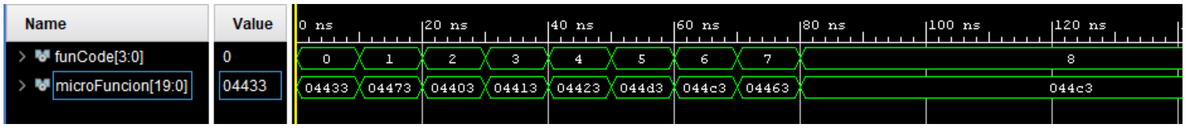
FORMAS DE ONDA

DECODIFICADOR



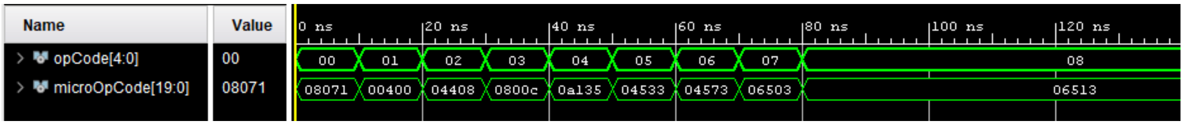
Name	Value
opCode[4:0]	12
TIPOR	0
BEQI	0
BNEQI	0
BLTI	0
BLETI	0
BGTI	0
BGETI	1

MEMORIA DE CÓDIGO DE FUNCIÓN

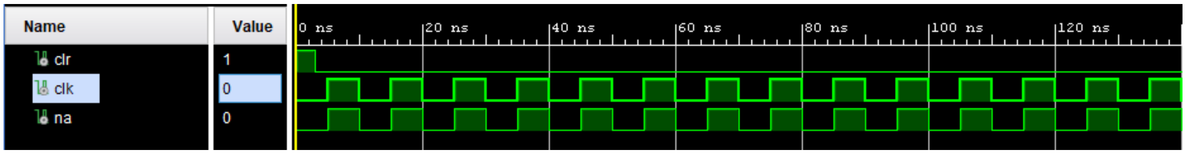


Name	Value
funCode[3:0]	0
microFuncion[19:0]	04433

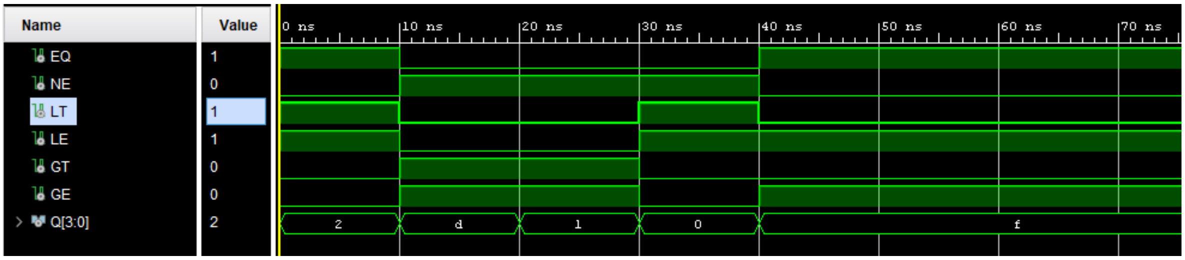
MEMORIA DE CÓDIGO DE OPERACIÓN



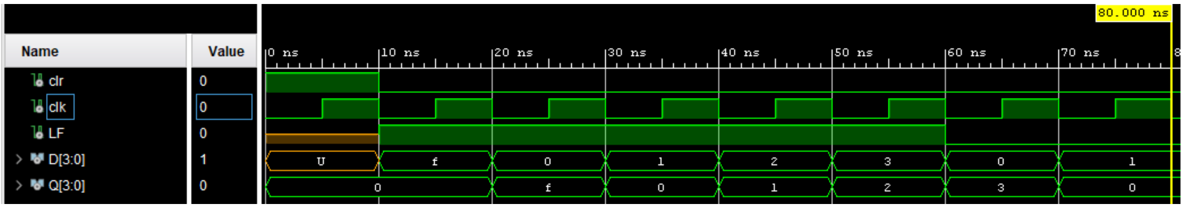
NIVEL



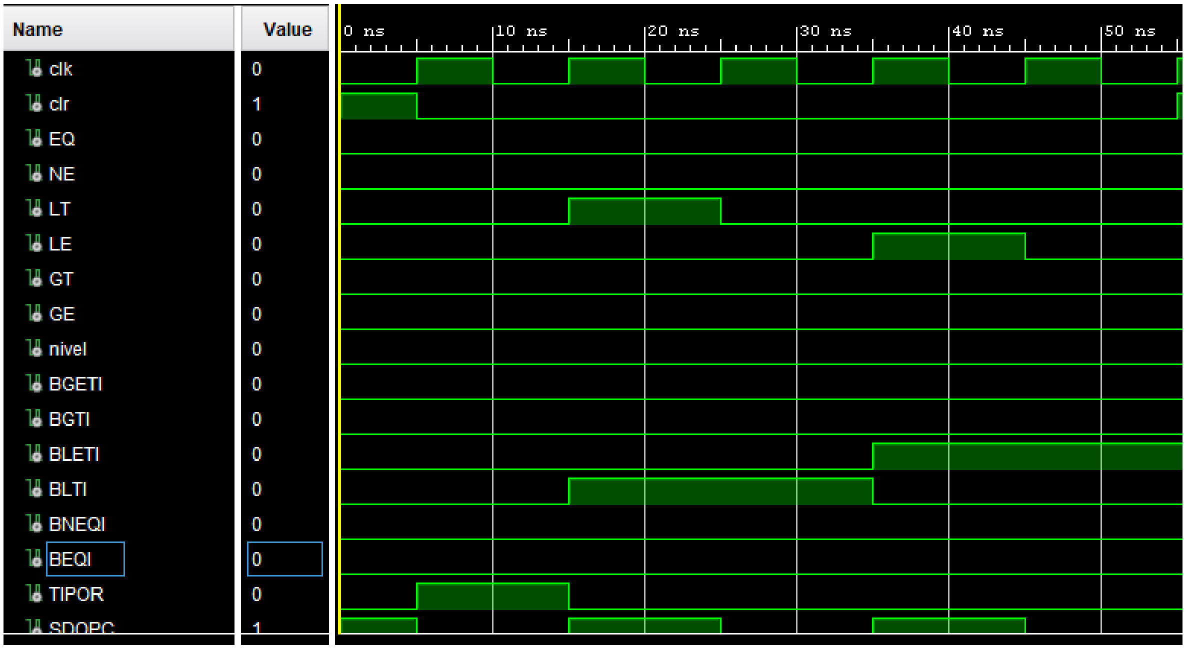
CONDICIÓN



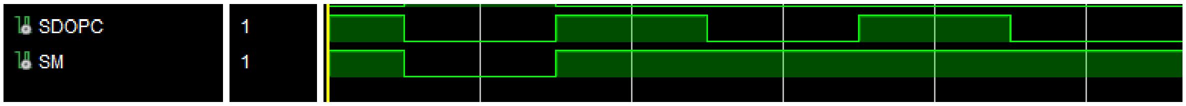
REGISTRO DE BANDERAS



CONTROL



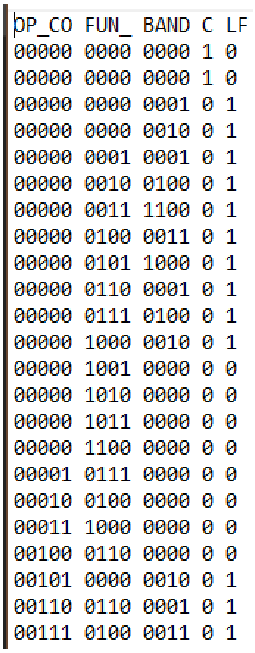
Name	Value
clk	0
clr	1
EQ	0
NE	0
LT	0
LE	0
GT	0
GE	0
nivel	0
BGETI	0
BGTI	0
BLETI	0
BLTI	0
BNEQI	0
BEQI	0
TIPOR	0
SDOPC	1



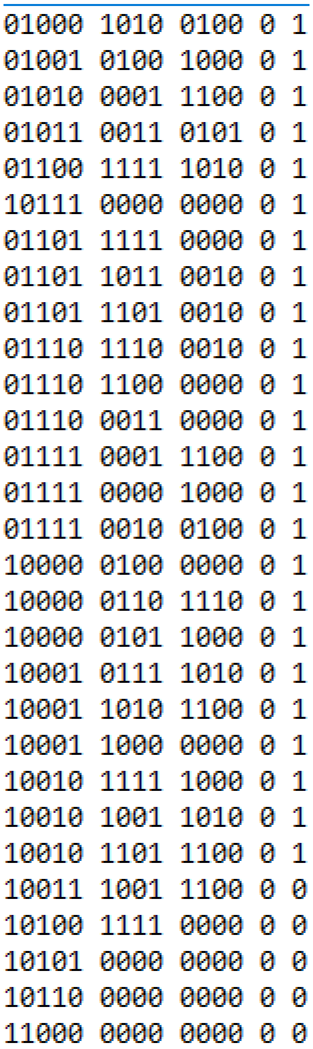
SDOPC	1
SM	1

ARCHIVOS

ENTRADA

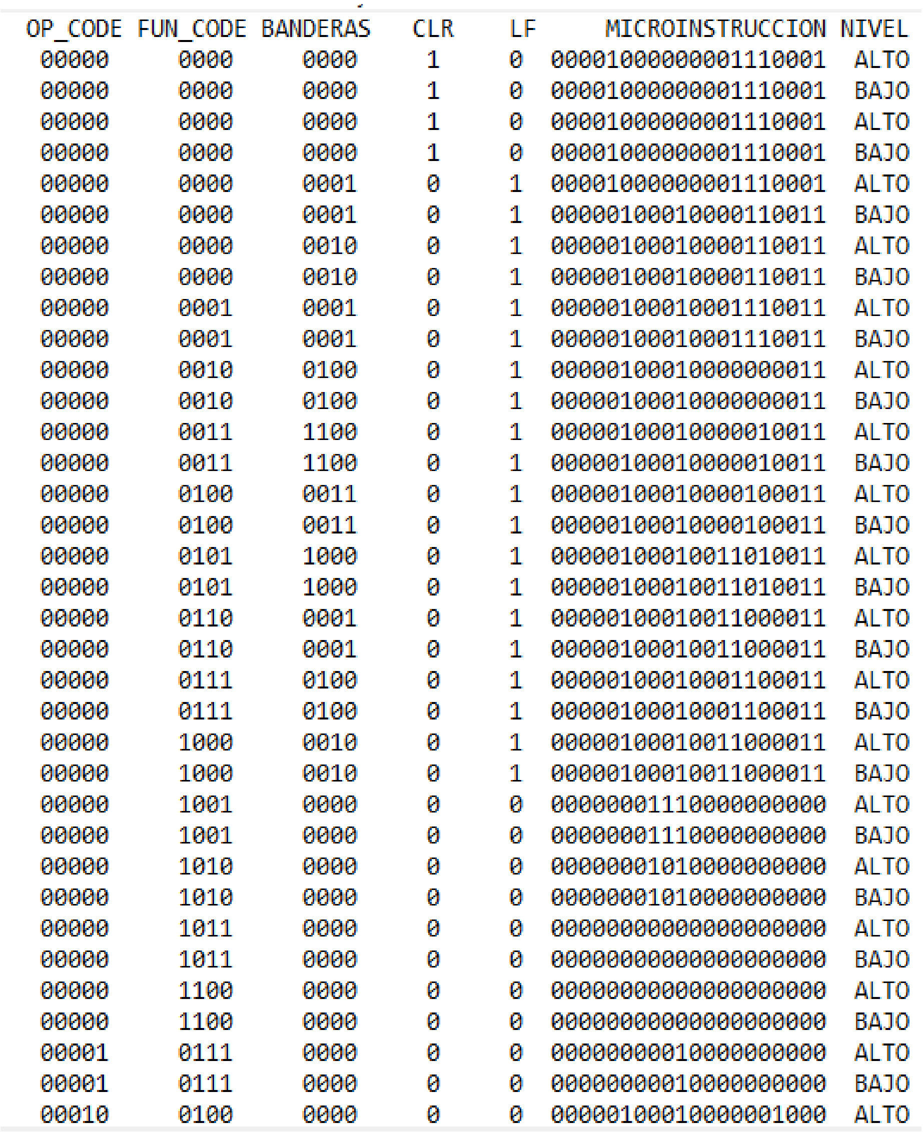


```
OP_CO FUN_ BAND C LF
00000 0000 0000 1 0
00000 0000 0000 1 0
00000 0000 0001 0 1
00000 0000 0010 0 1
00000 0001 0001 0 1
00000 0010 0100 0 1
00000 0011 1100 0 1
00000 0100 0011 0 1
00000 0101 1000 0 1
00000 0110 0001 0 1
00000 0111 0100 0 1
00000 1000 0010 0 1
00000 1001 0000 0 0
00000 1010 0000 0 0
00000 1011 0000 0 0
00000 1100 0000 0 0
00001 0111 0000 0 0
00010 0100 0000 0 0
00011 1000 0000 0 0
00100 0110 0000 0 0
00101 0000 0010 0 1
00110 0110 0001 0 1
00111 0100 0011 0 1
```

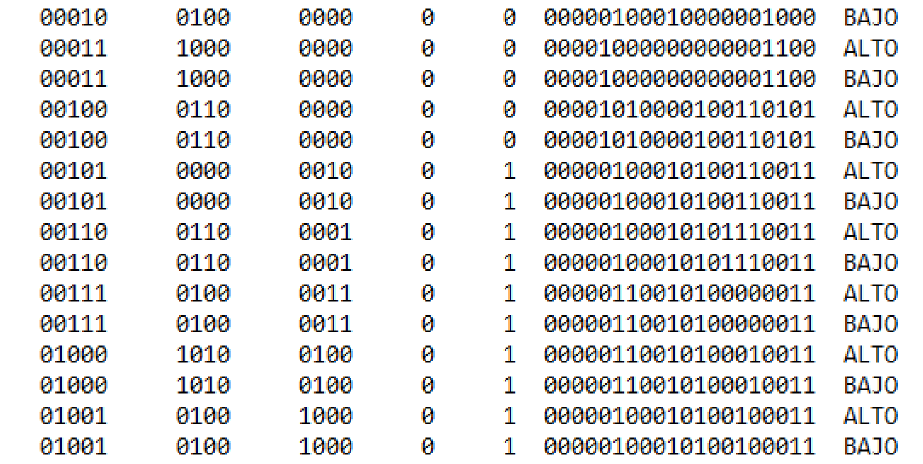


```
01000 1010 0100 0 1
01001 0100 1000 0 1
01010 0001 1100 0 1
01011 0011 0101 0 1
01100 1111 1010 0 1
10111 0000 0000 0 1
01101 1111 0000 0 1
01101 1011 0010 0 1
01101 1101 0010 0 1
01110 1110 0010 0 1
01110 1100 0000 0 1
01110 0011 0000 0 1
01111 0001 1100 0 1
01111 0000 1000 0 1
01111 0010 0100 0 1
10000 0100 0000 0 1
10000 0110 1110 0 1
10000 0101 1000 0 1
10001 0111 1010 0 1
10001 1010 1100 0 1
10001 1000 0000 0 1
10010 1111 1000 0 1
10010 1001 1010 0 1
10010 1101 1100 0 1
10011 1001 1100 0 0
10100 1111 0000 0 0
10101 0000 0000 0 0
10110 0000 0000 0 0
11000 0000 0000 0 0
```

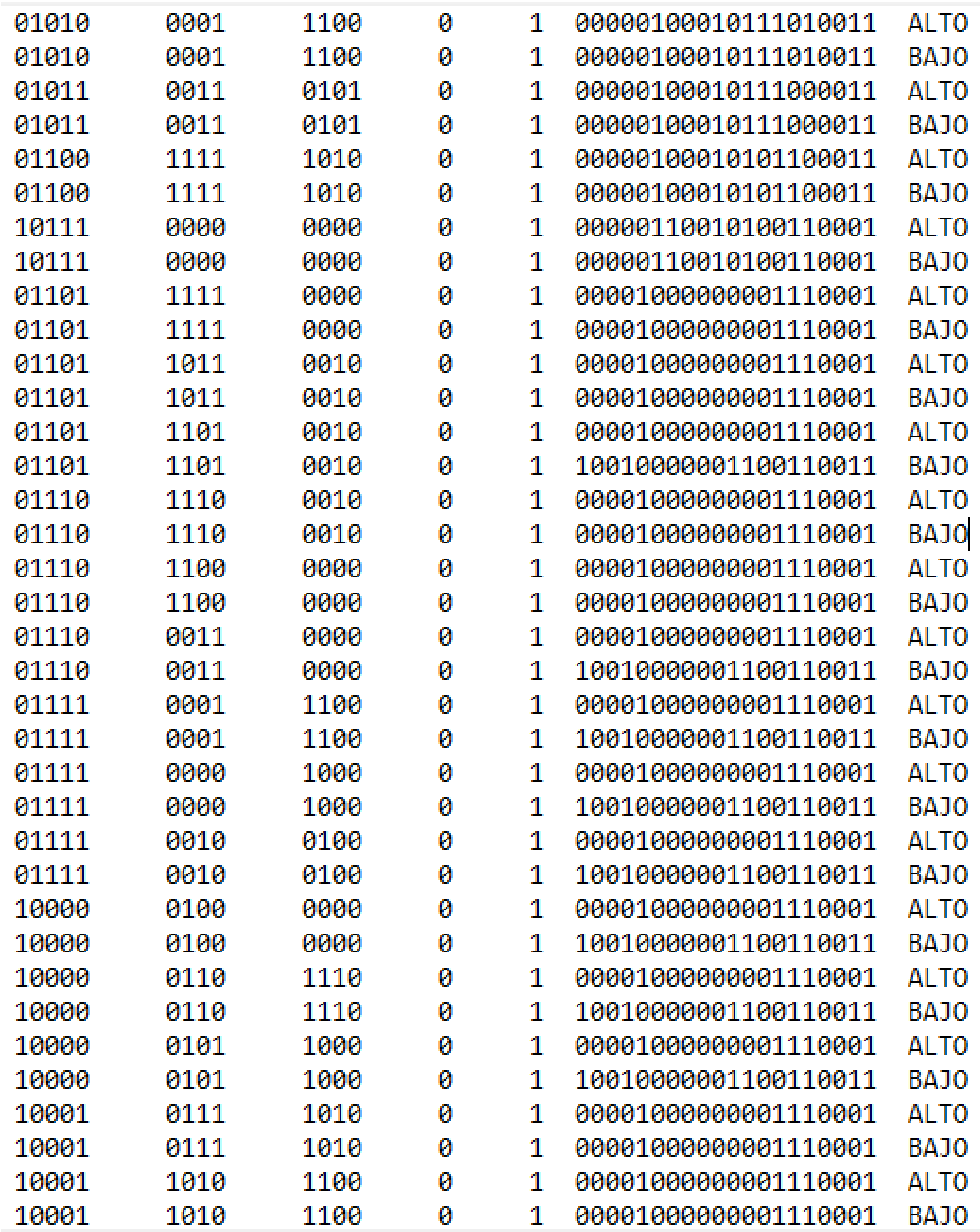
SALIDA



```
 OP_CODE FUN_CODE BANDERAS   CLR    LF     MICROINSTRUCCION NIVEL
  00000     0000     0000     1      0  00001000000001110001  ALTO
  00000     0000     0000     1      0  00001000000001110001  BAJO
  00000     0000     0000     1      0  00001000000001110001  ALTO
  00000     0000     0000     1      0  00001000000001110001  BAJO
  00000     0000     0001     0      1  00001000000001110001  ALTO
  00000     0000     0001     0      1  00000100010000110011  BAJO
  00000     0000     0010     0      1  00000100010000110011  ALTO
  00000     0000     0010     0      1  00000100010000110011  BAJO
  00000     0001     0001     0      1  00000100010001110011  ALTO
  00000     0001     0001     0      1  00000100010001110011  BAJO
  00000     0010     0100     0      1  00000100010000000011  ALTO
  00000     0010     0100     0      1  00000100010000000011  BAJO
  00000     0011     1100     0      1  00000100010000010011  ALTO
  00000     0011     1100     0      1  00000100010000010011  BAJO
  00000     0100     0011     0      1  00000100010000100011  ALTO
  00000     0100     0011     0      1  00000100010000100011  BAJO
  00000     0101     1000     0      1  00000100010011010011  ALTO
  00000     0101     1000     0      1  00000100010011010011  BAJO
  00000     0110     0001     0      1  00000100010011000011  ALTO
  00000     0110     0001     0      1  00000100010011000011  BAJO
  00000     0111     0100     0      1  00000100010001100011  ALTO
  00000     0111     0100     0      1  00000100010001100011  BAJO
  00000     1000     0010     0      1  00000100010011000011  ALTO
  00000     1000     0010     0      1  00000100010011000011  BAJO
  00000     1001     0000     0      0  00000001110000000000  ALTO
  00000     1001     0000     0      0  00000001110000000000  BAJO
  00000     1010     0000     0      0  00000001010000000000  ALTO
  00000     1010     0000     0      0  00000001010000000000  BAJO
  00000     1011     0000     0      0  00000000000000000000  ALTO
  00000     1011     0000     0      0  00000000000000000000  BAJO
  00000     1100     0000     0      0  00000000000000000000  ALTO
  00000     1100     0000     0      0  00000000000000000000  BAJO
  00001     0111     0000     0      0  00000000010000000000  ALTO
  00001     0111     0000     0      0  00000000010000000000  BAJO
  00010     0100     0000     0      0  00000100010000001000  ALTO
```



```
  00010     0100     0000     0      0  00000100010000001000  BAJO
  00011     1000     0000     0      0  00001000000000001100  ALTO
  00011     1000     0000     0      0  00001000000000001100  BAJO
  00100     0110     0000     0      0  00001010000100110101  ALTO
  00100     0110     0000     0      0  00001010000100110101  BAJO
  00101     0000     0010     0      1  00000100010100110011  ALTO
  00101     0000     0010     0      1  00000100010100110011  BAJO
  00110     0110     0001     0      1  00000100010101110011  ALTO
  00110     0110     0001     0      1  00000100010101110011  BAJO
  00111     0100     0011     0      1  00000110010100000011  ALTO
  00111     0100     0011     0      1  00000110010100000011  BAJO
  01000     1010     0100     0      1  00000110010100010011  ALTO
  01000     1010     0100     0      1  00000110010100010011  BAJO
  01001     0100     1000     0      1  00000100010100100011  ALTO
  01001     0100     1000     0      1  00000100010100100011  BAJO
```



01010	0001	1100	0	1	00000100010111010011	ALTO
01010	0001	1100	0	1	00000100010111010011	BAJO
01011	0011	0101	0	1	00000100010111000011	ALTO
01011	0011	0101	0	1	00000100010111000011	BAJO
01100	1111	1010	0	1	00000100010101100011	ALTO
01100	1111	1010	0	1	00000100010101100011	BAJO
10111	0000	0000	0	1	00000110010100110001	ALTO
10111	0000	0000	0	1	00000110010100110001	BAJO
01101	1111	0000	0	1	00001000000001110001	ALTO
01101	1111	0000	0	1	00001000000001110001	BAJO
01101	1011	0010	0	1	00001000000001110001	ALTO
01101	1011	0010	0	1	00001000000001110001	BAJO
01101	1101	0010	0	1	00001000000001110001	ALTO
01101	1101	0010	0	1	10010000001100110011	BAJO
01110	1110	0010	0	1	00001000000001110001	ALTO
01110	1110	0010	0	1	00001000000001110001	BAJO
01110	1100	0000	0	1	00001000000001110001	ALTO
01110	1100	0000	0	1	00001000000001110001	BAJO
01110	0011	0000	0	1	00001000000001110001	ALTO
01110	0011	0000	0	1	10010000001100110011	BAJO
01111	0001	1100	0	1	00001000000001110001	ALTO
01111	0001	1100	0	1	10010000001100110011	BAJO
01111	0000	1000	0	1	00001000000001110001	ALTO
01111	0000	1000	0	1	10010000001100110011	BAJO
01111	0010	0100	0	1	00001000000001110001	ALTO
01111	0010	0100	0	1	10010000001100110011	BAJO
10000	0100	0000	0	1	00001000000001110001	ALTO
10000	0100	0000	0	1	10010000001100110011	BAJO
10000	0110	1110	0	1	00001000000001110001	ALTO
10000	0110	1110	0	1	10010000001100110011	BAJO
10000	0101	1000	0	1	00001000000001110001	ALTO
10000	0101	1000	0	1	10010000001100110011	BAJO
10001	0111	1010	0	1	00001000000001110001	ALTO
10001	0111	1010	0	1	00001000000001110001	BAJO
10001	1010	1100	0	1	00001000000001110001	ALTO
10001	1010	1100	0	1	00001000000001110001	BAJO

```
10010     1111     1000     0     1  00001000000001110001  ALTO
10010     1111     1000     0     1  00001000000001110001  BAJO
10010     1001     1010     0     1  00001000000001110001  ALTO
10010     1001     1010     0     1  00001000000001110001  BAJO
10010     1101     1100     0     1  00001000000001110001  ALTO
10010     1101     1100     0     1  00001000000001110001  BAJO
10011     1001     1100     0     0  00010000000000000000  ALTO
10011     1001     1100     0     0  00010000000000000000  BAJO
10100     1111     0000     0     0  01010000000000000000  ALTO
10100     1111     0000     0     0  01010000000000000000  BAJO
10101     0000     0000     0     0  00100000000000000000  ALTO
10101     0000     0000     0     0  00100000000000000000  BAJO
10110     0000     0000     0     0  00000000000000000000  ALTO
10110     0000     0000     0     0  00000000000000000000  BAJO
11000     0000     0000     0     0  00000000000000000000  ALTO
11000     0000     0000     0     0  00000000000000000000  BAJO
```