



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO



U.A: Arquitectura de Computadoras

“Práctica 13: Unidad de Control”

Grupo: 3CV11

Profesor: Vega García Nayeli

Sánchez Becerra Ernesto Daniel

Código de implementación MFunCode

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

entity MFunCode is
    generic ( n : integer := 4 );
    Port (
        codigo_funcion : in STD_LOGIC_VECTOR(n-1 downto 0);
        microinstruccion_fcode : out STD_LOGIC_VECTOR (19 downto 0)
    );
end MFunCode;

architecture Behavioral of MFunCode is
    -- Declaracion de microinstrucciones (Solo tipo R)
    -- SDMP UP DW WPC SR2 SWD SEXT SHE DIR WR SOP1 SOP2 ALUOP3 ALUOP2 ALUOP1 ALUOP0 SDMD WD SR LF

    constant R_ADD : STD_LOGIC_VECTOR (19 downto 0) := "00000100010000110011"; -- ADD 0
    constant R_SUB : STD_LOGIC_VECTOR (19 downto 0) := "00000100010001110011"; -- SUB 1

    constant R_AND : STD_LOGIC_VECTOR (19 downto 0) := "00000100010000000011"; -- AND 2
    constant R_OR : STD_LOGIC_VECTOR (19 downto 0) := "00000100010000010011"; -- OR 3
    constant R_XOR : STD_LOGIC_VECTOR (19 downto 0) := "00000100010000100011"; -- XOR 4
    constant R_NAND : STD_LOGIC_VECTOR (19 downto 0) := "00000100010011010011"; -- NAND 5
    constant R_NOR : STD_LOGIC_VECTOR (19 downto 0) := "00000100010011000011"; -- NOR 6
    constant R_XNOR : STD_LOGIC_VECTOR (19 downto 0) := "00000100010001110011"; -- XNOR 7
    constant R_NOT : STD_LOGIC_VECTOR (19 downto 0) := "00000100010011010011"; -- NOT 8

    constant R_SLL : STD_LOGIC_VECTOR (19 downto 0) := "00000001110000000000"; -- SLL 9
    constant R_SRL : STD_LOGIC_VECTOR (19 downto 0) := "00000001010000000000"; -- SRL 10

    type memoria is array (0 to (2*n)-1) of std_logic_vector(19 downto 0);

    constant funCode : memoria := (
        R_ADD,
        R_SUB,
        R_AND,
        R_OR,
        R_XOR,
        R_NAND,
        R_NOR,
        R_XNOR,
        R_NOT,
        R_SLL,
        R_SRL,
        others => (others => '0')
    );

    begin
        microinstruccion_fcode <= funCode(conv_integer(codigo_funcion));
    end Behavioral;
```

Código de implementación MOpCode

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

entity MOpCode is
    generic ( n : integer := 5 );
    Port (
        codigo_operacion : in STD_LOGIC_VECTOR(n-1 downto 0);
        microinstruccion : out STD_LOGIC_VECTOR (19 downto 0)
    );
end MOpCode;

architecture Behavioral of MOpCode is
    -- Declaracion de microinstrucciones (Todas menos tipo R)
    -- SDMP UP DW WPC SR2 SWD SEXT SHE DIR WR SOP1 SOP2 ALUOP3 ALUOP2 ALUOP1 ALUOP0 SDMD WD SR LF

    constant VERIFICACION : STD_LOGIC_VECTOR(19 downto 0) := "00001000000001110001"; -- VERIFICACION 0
    constant LI : STD_LOGIC_VECTOR(19 downto 0) := "00000000010000000000"; -- LI 1
    constant LWI : STD_LOGIC_VECTOR(19 downto 0) := "00001100010000001000"; -- LWI 2
    constant SWI : STD_LOGIC_VECTOR(19 downto 0) := "00001000000000001100"; -- SWI 3
    constant SW : STD_LOGIC_VECTOR(19 downto 0) := "00001010000100110101"; -- SW 4

    constant ADDI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010100110011"; -- ADDI 5
    constant SUBI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010101110011"; -- SUBI 6

    constant ANDI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010100000011"; -- ANDI 7
    constant ORI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010100010011"; -- ORI 8
    constant XORI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010100100011"; -- XORI 9
    constant NANDI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010111010011"; -- NANDI 10
    constant NORI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010111000011"; -- NORI 11
    constant XNORI : STD_LOGIC_VECTOR(19 downto 0) := "0000010001011100011"; -- XNORI 12

    constant SALTO : STD_LOGIC_VECTOR(19 downto 0) := "10010000001100110011"; -- SALTO 13-18
    constant B : STD_LOGIC_VECTOR(19 downto 0) := "00010000000000000000"; -- B 19

    constant CALL : STD_LOGIC_VECTOR(19 downto 0) := "01010000000000000000"; -- CALL 20
    constant RET : STD_LOGIC_VECTOR(19 downto 0) := "00100000000000000000"; -- RET 21
    constant NOP : STD_LOGIC_VECTOR(19 downto 0) := "00000000000000000000"; -- NOP 22

    constant LW : STD_LOGIC_VECTOR(19 downto 0) := "00000110010100110001"; -- LW 23

    type memoria is array (0 to (2*n)-1) of std_logic_vector(19 downto 0);
    constant opCode : memoria := (
        VERIFICACION,
        LI,
        LWI,
        SWI,
        SW,
        ADDI,
        SUBI,
        ANDI,
        ORI,
        XORI,
        NANDI,
        NORI,
        XNORI,
        SALTO, -- BEQI
        SALTO, -- BNEI
        SALTO, -- BLTI
        SALTO, -- BLETI
        SALTO, -- BGTI
        SALTO, -- BGETI
        B,
        CALL,
        RET,
        NOP,
        LW,
        others => (others => '0')
    );
    begin
        microinstruccion <= opCode(conv_integer(codigo_operacion));
    end Behavioral;
```

Código de implementación Decodificador de Instrucción

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity decodificador is
    Port (
        codigo_operacion : in STD_LOGIC_VECTOR(4 downto 0);
        tipo_R : out STD_LOGIC;
        BEQI : out STD_LOGIC;
        BNEI : out STD_LOGIC;
        BLTI : out STD_LOGIC;
        BLETI : out STD_LOGIC;
        BGTI : out STD_LOGIC;
        BGETI : out STD_LOGIC
    );
end decodificador;
architecture Behavioral of decodificador is
    begin
        process(codigo_operacion)
        begin
            if(codigo_operacion = "00000") then -- TIPO R
                tipo_R <= '1';
                BEQI <= '0';
                BNEI <= '0';
                BLTI <= '0';
                BLETI <= '0';
                BGTI <= '0';
                BGETI <= '0';

            elsif(codigo_operacion = "01101") then -- BEQI
                tipo_R <= '0';
                BEQI <= '1';
                BNEI <= '0';
                BLTI <= '0';
                BLETI <= '0';
                BGTI <= '0';
                BGETI <= '0';

            elsif(codigo_operacion = "01110") then -- BNEI
                tipo_R <= '0';
                BEQI <= '0';
                BNEI <= '1';
                BLTI <= '0';
                BLETI <= '0';
                BGTI <= '0';
                BGETI <= '0';

            elsif(codigo_operacion = "01111") then -- BLTI
                tipo_R <= '0';
                BEQI <= '0';
                BNEI <= '0';
                BLTI <= '1';
                BLETI <= '0';
                BGTI <= '0';
                BGETI <= '0';

            elsif(codigo_operacion = "10000") then -- BLETI
                tipo_R <= '0';
                BEQI <= '0';
                BNEI <= '0';
                BLTI <= '0';
                BLETI <= '1';
                BGTI <= '0';
                BGETI <= '0';

            elsif(codigo_operacion = "10001") then -- BGTI
                tipo_R <= '0';
                BEQI <= '0';
                BNEI <= '0';
                BLTI <= '0';
                BLETI <= '0';
                BGTI <= '1';
                BGETI <= '0';

            elsif(codigo_operacion = "10010") then -- BGETI
                tipo_R <= '0';
                BEQI <= '0';
                BNEI <= '0';
                BLTI <= '0';
                BLETI <= '0';
                BGTI <= '0';
                BGETI <= '1';

            else -- NO ES TIPO R NI SALTO CONDICIONAL
                tipo_R <= '0';
                BEQI <= '0';
                BNEI <= '0';
                BLTI <= '0';
                BLETI <= '0';
                BGTI <= '0';
                BGETI <= '0';
            end if;
        end process;
    end Behavioral;
```

Código de implementación Unidad de Control

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

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

architecture Behavioral of control is
    type estados is (A);
    signal estado_actual, estado_siguiente : estados;

    begin

        trnasicion : process(clr, clk)-- establece el cambio de estado actual a estado siguiente
        begin
            if(clr = '1') then
                estado_actual <= A;
            elsif(rising_edge(clk)) then
                estado_actual <= estado_siguiente;
            end if;
        end process;

        asm : process(TIPOR, BEQI, BNEI, BLTI, BLETI, BGTI, BGETI, NA, EQ, NE, LT, LE, GT, GE, NA, estado_actual)
        begin
            SM <= '0';
            SDOPC <= '0';

            case estado_actual is
                when A =>
                    if(TIPOR = '1') then
                        SM <= '0';
                    else
                        if(BEQI = '0') then
                            if (BNEI = '0') then
                                if (BLTI = '0') then
                                    if(BLETI = '0') then
                                        if(BGTI = '0') then
                                            if(BGETI = '0') then
                                                SDOPC <= '1';
                                                SM <= '1';
                                                estado_siguiente <= A;
                                            else
                                                if(NA = '1') then
                                                    SDOPC <= '0';
                                                    SM <= '1';
                                                    estado_siguiente <= A;
                                                else
                                                    if(GE = '1') then
                                                        SDOPC <= '1';
                                                        SM <= '1';
                                                        estado_siguiente <= A;
                                                    else
                                                        SDOPC <= '0';
                                                        SM <= '1';
                                                        estado_siguiente <= A;
                                                    end if;
                                                end if;
                                            end if;
                                        end if;
                                    end if;
                                end if;
                            end if;
                        else
                            if(NA = '1') then
                                SDOPC <= '0';
                                SM <= '1';
                                estado_siguiente <= A;
                            else
                                if(GT = '1') then
                                    SDOPC <= '1';
                                    SM <= '1';
                                    estado_siguiente <= A;
                                else
                                    SDOPC <= '0';
                                    SM <= '1';
                                    estado_siguiente <= A;
                                end if;
                            end if;
                        end if;
                    end if;
                end if;
            end case;

            if(NA = '1') then
                SDOPC <= '0';
                SM <= '1';
                estado_siguiente <= A;
            else
                if(GT = '1') then
                    SDOPC <= '1';
                    SM <= '1';
                    estado_siguiente <= A;
                else
                    SDOPC <= '0';
                    SM <= '1';
                    estado_siguiente <= A;
                end if;
            end if;
        end process;
    end
end Behavioral;
```

else

```
    if(LE = '1') then
        SDOPC <= '1';
        SM <= '1';
        estado_siguiente <= A;
```

```
    else
        SDOPC <= '0';
        SM <= '1';
        estado_siguiente <= A;
    end if;
```

```
end if;
```

```
end if;
```

```
else
```

```
    if(NA = '1') then
        SDOPC <= '0';
        SM <= '1';
        estado_siguiente <= A;
```

```
    else
```

```
        if(LT = '1') then
            SDOPC <= '1';
            SM <= '1';
            estado_siguiente <= A;
```

```
        else
            SDOPC <= '0';
            SM <= '1';
            estado_siguiente <= A;
```

```
        end if;
```

```
    end if;
```

```
end if;
```

```
else
```

```
    if(NA = '1') then
        SDOPC <= '0';
        SM <= '1';
        estado_siguiente <= A;
```

```
    else
```

```
        if(NE = '1') then
            SDOPC <= '1';
            SM <= '1';
            estado_siguiente <= A;
```

```
        else
            SDOPC <= '0';
            SM <= '1';
            estado_siguiente <= A;
```

```
        end if;
```

```
    end if;
```

```
end if;
```

```
else
```

```
    if(NA = '1') then
        SDOPC <= '0';
        SM <= '1';
        estado_siguiente <= A;
```

```
    else
```

```
        if(EQ = '1') then
            SDOPC <= '1';
            SM <= '1';
            estado_siguiente <= A;
```

```
        else
            SDOPC <= '0';
            SM <= '1';
            estado_siguiente <= A;
```

```
        end if;
```

```
    end if;
```

```
end if;
```

```
end if;
```

```
end case;
```

```
end process;
```

```
end Behavioral;
```

Código de implementación Nivel

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

entity nivel is
    Port ( clk : in STD_LOGIC;
          clr : in STD_LOGIC;
          na : out STD_LOGIC);
end nivel;

architecture Behavioral of nivel is
    signal pclk, nclk : STD_LOGIC;
begin

    ALTO : process(clr, clk)
    begin
        if(clr = '1') then
            pclk <= '0';
        elsif(rising_edge(clk)) then
            pclk <= not pclk;
        end if;
    end process;

    BAJ0 : process(clr, clk)
    begin
        if(clr = '1') then
            nclk <= '0';
        elsif(falling_edge(clk)) then
            nclk <= not nclk;
        end if;
    end process;

    na <= nclk xor pclk; -- LOGICA COMBINATORIA

end Behavioral;
```

Código de implementación Condición

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

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

architecture Behavioral of condicion is
    -- BANDERAS : 3 - OV, 2- N, 1 - Z, 0 - C
    signal C, Z, N, OV : STD_LOGIC;

    begin

        -- fetch de banderas
        C <= banderas(0);
        Z <= banderas(1);
        N <= banderas(2);
        OV <= banderas(3);

        EQ <= Z;
        NE <= not Z;
        LT <= not C;
        LE <= Z or (not C);
        GT <= (not Z) and C;
        GE <= C;

    end Behavioral;
```


Código de implementación Mux20bits

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

entity mux20bits is
    Port (
        codigo_fu: in STD_LOGIC_VECTOR(19 downto 0);
        codigo_op: in STD_LOGIC_VECTOR(19 downto 0);
        sm : in STD_LOGIC;
        salida : out STD_LOGIC_VECTOR(19 downto 0)
    );
end mux20bits;

architecture Behavioral of mux20bits is
    begin

        with sm select
            salida <= codigo_op when '1',
                    codigo_fu when others;
    end Behavioral;
```

Código de implementación Mux5bits

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

entity mux5bits is
    Port (
        codigo_op: in STD_LOGIC_VECTOR(4 downto 0);
        sdopc : in STD_LOGIC;
        salida : out STD_LOGIC_VECTOR(4 downto 0)
    );
end mux5bits;

architecture Behavioral of mux5bits is
    constant cero : STD_LOGIC_VECTOR(4 downto 0) := "00000";
begin
    with sdopc select
        salida <=
            codigo_op when '1',
            cero when others;

end Behavioral;
```

Código de implementación Registro

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

entity registro is
    Port (
        banderas_entrada : in STD_LOGIC_VECTOR(3 downto 0);
        lf : in STD_LOGIC;
        clk : in STD_LOGIC;
        clr : in STD_LOGIC;
        banderas_salida : out STD_LOGIC_VECTOR(3 downto 0)
    );
end registro;

architecture Behavioral of registro is
begin

    process (clr, clk)
    begin
        if(clr = '1') then
            banderas_salida <= "0000";
        elsif(falling_edge(clk)) then
            if(lf = '1')then
                banderas_salida <= banderas_entrada;
            end if;
        end if;
    end process;
end Behavioral;
```

Código de implementación Arquitectura Completa

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

entity uniddad_control is
    Port (
        funCode : in STD_LOGIC_VECTOR(3 downto 0);
        opCode : in STD_LOGIC_VECTOR(4 downto 0);
        clk, clr, lf : in STD_LOGIC;
        banderas : in STD_LOGIC_VECTOR(3 downto 0);
        microInstruccion : out STD_LOGIC_VECTOR(19 downto 0)
    );
end uniddad_control;

architecture Behavioral of uniddad_control is

    -- memoria de codigo de funcion
    component MFunCode is
        Port (
            codigo_funcion : in STD_LOGIC_VECTOR(3 downto 0);
            microinstruccion_fcode : out STD_LOGIC_VECTOR (19 downto 0));
    end component;

    -- memoria de codigo de operacion
    component MOpCode is
        Port (
            codigo_operacion : in STD_LOGIC_VECTOR(4 downto 0);
            microinstruccion : out STD_LOGIC_VECTOR (19 downto 0));
    end component;

    -- condicion
    component condicion is
        Port(
            banderas : in STD_LOGIC_VECTOR(3 downto 0);
            EQ, NE, LT, LE, GT, GE : out STD_LOGIC);
    end component;

    -- control (ASM)
    component control is
        Port (
            TIPOR, BEQI, BNEI, BLTI, BLETI, BGTI, BGETI : in STD_LOGIC;
            EQ, NE, LT, LE, GT, GE : in STD_LOGIC;
            clk, clr, NA: in STD_LOGIC;
            SDOPC, SM : out STD_LOGIC);
    end component;

    -- decodificador
    component decodificador is
        Port (
            codigo_operacion : in STD_LOGIC_VECTOR(4 downto 0);
            tipo_R : out STD_LOGIC;
            BEQI ,BNEI , BLTI, BLETI, BGTI, BGETI : out STD_LOGIC);
    end component;

    -- multiplexor de 5 bits
    component mux5bits is
        Port (
            codigo_op: in STD_LOGIC_VECTOR(4 downto 0);
            sdopc : in STD_LOGIC;
            salida : out STD_LOGIC_VECTOR(4 downto 0));
    end component;
```

```

-- multiplexor de 20 bits
component mux20bits is
  Port (
    codigo_fu: in STD_LOGIC_VECTOR(19 downto 0);
    codigo_op: in STD_LOGIC_VECTOR(19 downto 0);
    sm : in STD_LOGIC;
    salida : out STD_LOGIC_VECTOR(19 downto 0));
end component;

-- nivel
component nivel is
  Port ( clk : in STD_LOGIC;
        clr : in STD_LOGIC;
        na : out STD_LOGIC);
end component;

-- registro
component registro is
  Port (
    banderas_entrada : in STD_LOGIC_VECTOR(3 downto 0);
    lf, clk, clr : in STD_LOGIC;
    banderas_salida : out STD_LOGIC_VECTOR(3 downto 0));
end component;

-- declaracion de señales de transporte (BUSES)
signal EQ, NE, LT, LE, GT, GE : STD_LOGIC;
signal NA, SDOPC, SM : STD_LOGIC;
signal TIPOR, BEQI, BNEI, BLTI, BLETI, BGTI, BGETI: STD_LOGIC;
signal auxUFCcode, auxUOpCode, auxSalida : STD_LOGIC_VECTOR(19 downto 0);
signal auxOpCode : STD_LOGIC_VECTOR(4 downto 0);
signal auxBanderas : STD_LOGIC_VECTOR(3 downto 0);

begin

-- instanciar y mapear modulo de memoria de codigo de funcion
MFC : MFunCode
  Port map (
    codigo_function => funCode,
    microInstruccion_fcode => auxUFCcode
  );

-- instanciar y mapear modulo de memoria de codigo de operacion
MOC : MOpCode
  Port map(
    codigo_operacion => auxOpCode,
    microInstruccion => auxUOpCode
  );

-- instanciar y mapear modulo de condicion
COND : condicion
  Port map(
    banderas => auxBanderas,
    EQ => EQ,
    NE => NE,
    LT => LT,
    LE => LE,
    GT => GT,
    GE => GE
  );

```

```

-- Instanciar y mapear modulo de control
CONTR : component control
  Port map(
    TIPOR => TIPOR,
    BEQI => BEQI,
    BNEI => BNEI,
    BLTI => BLTI,
    BLETI => BLETI,
    BGTI => BGTI,
    BGETI => BGETI,
    EQ => EQ,
    NE => NE,
    LT => LT,
    LE => LE,
    GT => GT,
    GE => GE,
    clk => clk,
    clr => clr,
    NA => NA,
    SDOPC => SDOPC,
    SM => SM
  );

-- Instanciar y mapear modulo de decodificacion
DECO : decodificador
  Port map(
    codigo_operacion => opCode,
    tipo_R => TIPOR,
    BEQI => BEQI,
    BNEI => BNEI,
    BLTI => BLTI,
    BLETI => BLETI,
    BGTI => BGTI,
    BGETI => BGETI
  );

-- Instanciar y mapear mux de 5 bits
MUX5 : Mux5bits
  Port map(
    codigo_op => opCode,
    sdopc => SDOPC,
    salida => auxOpCode
  );

-- Instanciar y mapear mux de 20 bits
MUX20 : mux20bits
  Port map(
    codigo_fu => auxUFCCode,
    codigo_op => auxUOpCode,
    salida => auxSalida,
    sm => SM
  );

-- Instanciar y mapear el modulo nivel
NIV : nivel
  Port map(
    clk => clk,
    clr => clk,
    na => NA
  );

-- Instanciar y mapear el modulo de registro
REG : registro
  Port map(
    banderas_entrada => banderas,
    lf => lf,
    clk => clk,
    clr => clr,
    banderas_salida => auxBanderas
  );

microInstruccion <= auxSalida;

end Behavioral;

```

Código de simulación Arquitectura Completa

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE IEEE.STD_LOGIC_unsigned.ALL;
LIBRARY STD;
USE STD.TEXTIO.ALL;
USE ieee.std_logic_TEXTIO.ALL;

entity tb_uniddad_control is
end tb_uniddad_control;

architecture Behavioral of tb_uniddad_control is
    component uniddad_control is
        Port (
            funCode : in STD_LOGIC_VECTOR(3 downto 0);
            opCode : in STD_LOGIC_VECTOR(4 downto 0);
            clk, clr, lf : in STD_LOGIC;
            banderas : in STD_LOGIC_VECTOR(3 downto 0);
            microInstruccion : out STD_LOGIC_VECTOR(19 downto 0)
        );
    end component;

    -- señales de comunicacion (Buses)
    signal funCode : STD_LOGIC_VECTOR(3 downto 0);
    signal opCode : STD_LOGIC_VECTOR(4 downto 0);
    signal clk, clr, lf : STD_LOGIC;
    signal banderas : STD_LOGIC_VECTOR(3 downto 0);
    signal microInstruccion : STD_LOGIC_VECTOR(19 downto 0);

begin

    uut : uniddad_control Port map(
        funCode => funCode,
        opCode => opCode,
        clk => clk,
        clr => clr,
        lf => lf,
        banderas => banderas,
        microInstruccion => microInstruccion
    );

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

    -- Estimulos de proceso
    SIM : process
        -- ENTRADAS
        file ENTRADAS : TEXT;
        variable LINEA_E : line;

        variable V_OP_CODE : STD_LOGIC_VECTOR(4 downto 0);
        variable V_FUN_CODE : STD_LOGIC_VECTOR(3 downto 0);
        variable V_BANDERAS : STD_LOGIC_VECTOR(3 downto 0);
        variable V_CLR : STD_LOGIC;
        variable V_LF : STD_LOGIC;

        variable CADENA : STRING(1 TO 9);
        variable CADENA2 : STRING(1 TO 21);

        -- SALIDAS
        file SALIDAS : TEXT;
        variable LINEA_RES : line;

        variable V_MICROINSTRUCCION : STD_LOGIC_VECTOR(19 downto 0);

    begin
        file_open(SALIDAS, "C:\Users\Aaron\Desktop\P14\SALIDAS.txt", WRITE_MODE);
        file_open(ENTRADAS, "C:\Users\Aaron\Desktop\P14\ENTRADAS.txt", READ_MODE);

        --Encabezados
        CADENA := "OP_CODE ";
        write(LINEA_RES, CADENA, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "OP_CODE"
        CADENA := "FUN_CODE ";
        write(LINEA_RES, CADENA, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "FUN_CODE"
        CADENA := "BANDERAS ";
        write(LINEA_RES, CADENA, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "BANDERAS"
        CADENA := "CLR ";
        write(LINEA_RES, CADENA, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "CLR"
        CADENA := "LF ";
        write(LINEA_RES, CADENA, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "LF"
        CADENA2 := "MICROINSTRUCCION ";
        write(LINEA_RES, CADENA2, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "MICROINSTRUCCION"
        CADENA := "NIVEL ";
        write(LINEA_RES, CADENA, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "NIVEL"

        writeline(SALIDAS, LINEA_RES);-- escribe la linea en el archivo
```

```

-- Leer y escribir estímulos
clr <= '1';
wait for 10 ns;
clr <= '0';
wait for 10 ns;

for i in 1 to 52 loop

    -- lee la línea completa
    readline(ENTRADAS, LINEA_E);
    -- leer operación code
    read(LINEA_E, V_OP_CODE);
    -- leer función code
    read(LINEA_E, V_FUN_CODE);
    -- BANDERAS
    read(LINEA_E, V_BANDERAS);
    -- CLR
    read(LINEA_E, V_CLR);
    -- LF
    read(LINEA_E, V_LF);

    opCode <= V_OP_CODE;
    funCode <= V_FUN_CODE;
    banderas <= V_BANDERAS;
    lf <= V_LF;

    wait until falling_edge(clk);

    clr <= V_CLR;
    wait for 2 ns;

    CADENA := "ALTO    ";
    V_MICROINSTRUCCION := microInstruccion;

    write(LINEA_RES, V_OP_CODE, LEFT, 10);
    write(LINEA_RES, V_FUN_CODE, LEFT, 10);
    write(LINEA_RES, V_BANDERAS, LEFT, 10);
    write(LINEA_RES, V_CLR, LEFT, 10);
    write(LINEA_RES, V_LF, LEFT, 10);
    write(LINEA_RES, V_MICROINSTRUCCION, LEFT, 21);
    write(LINEA_RES, CADENA, LEFT, 10);
    writeline(SALIDAS, LINEA_RES);

    wait until rising_edge(clk);

    CADENA := "BAJO    ";
    wait for 2 ns;

    V_MICROINSTRUCCION := microInstruccion;

    write(LINEA_RES, V_OP_CODE, LEFT, 10);
    write(LINEA_RES, V_FUN_CODE, LEFT, 10);
    write(LINEA_RES, V_BANDERAS, LEFT, 10);
    write(LINEA_RES, V_CLR, LEFT, 10);
    write(LINEA_RES, V_LF, LEFT, 10);
    write(LINEA_RES, V_MICROINSTRUCCION, LEFT, 21);
    write(LINEA_RES, CADENA, LEFT, 10);
    writeline(SALIDAS, LINEA_RES);

    -- CADENA := "          ";
    -- write(LINEA_RES, CADENA, LEFT, 10);
    -- writeline(SALIDAS, LINEA_RES);

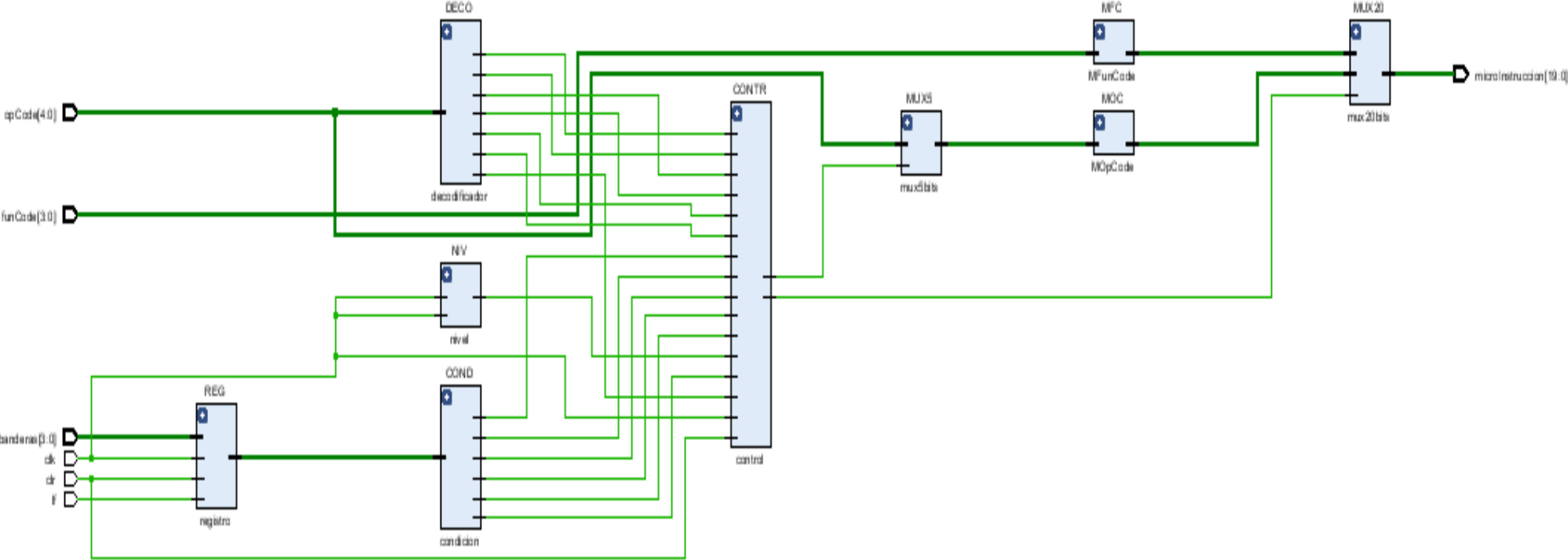
end loop;

-- cierra el archivo
file_close(SALIDAS);
file_close(ENTRADAS);

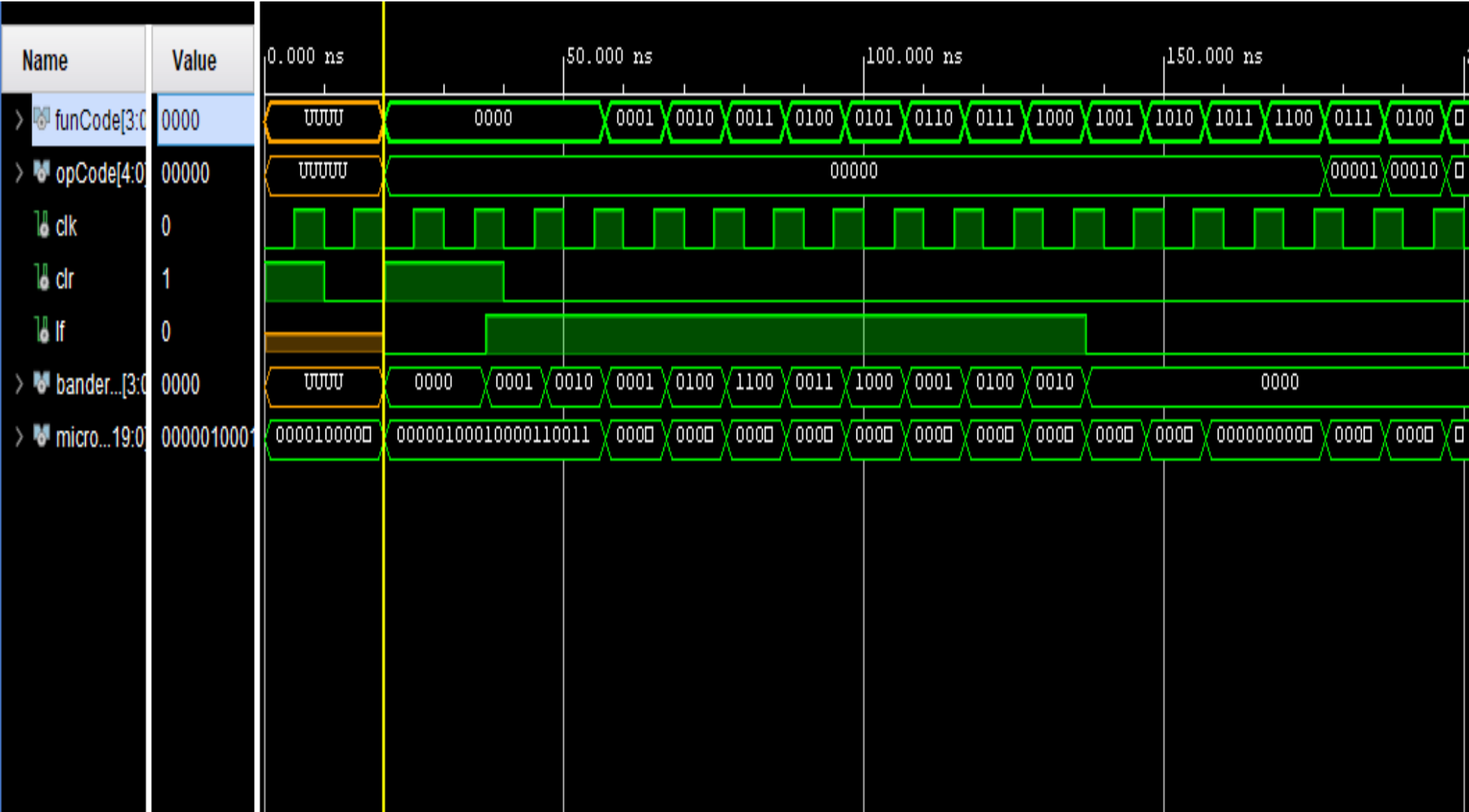
wait;
end process;
end Behavioral;

```



Diagrama RTL Arquitectura Completa



Forma de onda Arquitectura Completa




Archivo de entradas

 ENTRADAS: Bloc de notas

Archivo	Edición	Formato	Ver	Ayuda
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
00000	0111	0000	0	0

Archivo de salidas

 SALIDAS: Bloc de notas

Archivo	Edición	Formato	Ver	Ayuda		
OP_CODE	FUN_CODE	BANDERAS	CLR	LF	MICROINSTRUCCION	NIVEL
00000	0000	0000	1	0	00000100010000110011	ALTO
00000	0000	0000	1	0	00000100010000110011	BAJO
00000	0000	0000	1	0	00000100010000110011	ALTO
00000	0000	0000	1	0	00000100010000110011	BAJO
00000	0000	0001	0	1	00000100010000110011	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	000001000100000000011	ALTO
00000	0010	0100	0	1	000001000100000000011	BAJO
00000	0011	1100	0	1	000001000100000010011	ALTO
00000	0011	1100	0	1	000001000100000010011	BAJO
00000	0100	0011	0	1	00000100010000100011	ALTO
00000	0100	0011	0	1	00000100010000100011	BAJO